micro processors 2

samenstelling ir. M. P. J. Stevens en
ir. J. Wilmink

Kluwer Technische Boeken B.V. Deventer — Antwerpen
Stichting Teleac - Utrecht

Woord vooraf

Toen de Stichting Teleac in het najaar van 1978 begon met de uitzending van de cursus *Microprocessors 1*, bestond er geen vermoeden over de mogelijke belangstelling voor deze cursus. De oorspronkelijke verwachting van enkele duizenden werd flink overtroffen. Meer dan 10 000 cursisten schreven in op de eerste uitzending van deze cursus. Deze cursus is inmiddels ook als boek uitgegeven.
Mede op basis van de ervaringen, opgedaan met de cursus Microprocessors 1 is de cursus Microprocessors 2 ontwikkeld. Dit is op de eerste plaats een software-cursus.
Wij willen de lezer leren een programma in de taal BASIC te schrijven. Daarnaast worden een aantal hoofdstukken gewijd aan een machinetaal: de assembleertaal.
Ook de mogelijkheden voor het uitbreiden van systemen komen aan de orde. Hoewel in de tekst de Teleac/DAI-computer als voorbeeld is genomen, is dit boek ook bruikbaar bij andere microcomputers.

Ook voor de cursus Microprocessors 2 was de belangstelling opvallend groot. Een aantal van 10 000 cursisten schreef in voor deze software-cursus. Voor deze gebonden Kluwer-uitgave werden alle commentaren verwerkt die we tot voorjaar 1980 ontvingen.
We hebben de indruk dat we een zinvolle en positieve bijdrage hebben geleverd aan de discussie over de introductie van de micro-elektronica en willen met deze boekuitgaven aan de blijvende vraag naar ons cursusmateriaal voldoen.

Drs. Chriet Titulaer
Hoofd afd. schriftelijk onderwijs van Teleac

Inhoud

Hoofdstuk 1

Inleiding

Om een computer efficiënt te kunnen gebruiken, is het gewenst dat men globaal weet hoe zo'n machine is opgebouwd. Programma's, die een computer uitvoert, bestaan uit een reeks van basisopdrachten. Om de structuur van deze programma's overzichtelijk te maken, is het nuttig de methode van het oplossen van het probleem met behulp van een zogenaamd stroomdiagram te beschrijven. Het programma kan daarna aan de hand van dit schema worden gecodeerd.

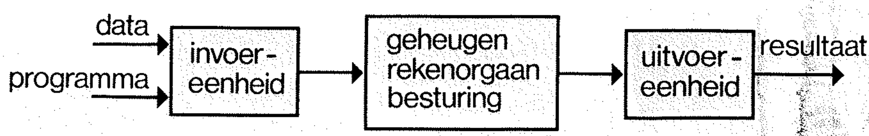
1.1. Het begrip computer

De computer kan het werk van de mens op vele terreinen vereenvoudigen. Het begrip 'computer' komt uit de Engelse taal en betekent: *rekenaar* of *rekenmachine*. In de eerste jaren van zijn bestaan was 'rekenen' dan ook de hoofdtaak van de computer. Nu is dit echter sterk veranderd. Vandaag de dag wordt bijvoorbeeld het verkeer geregeld door een verkeerscomputer. In een chemische fabriek neemt de computer het besturen van het chemische proces voor zijn rekening. We moeten ook niet vergeten dat de computer vaak binnendringt in de huizen van privépersonen in de vorm van de *personal* (eigen) *computer*.
Eigenlijk zouden we de betekenis van het begrip 'computer' moeten veranderen in: *data-verwerkend systeem* (data = gegevens of informatie). Dit houdt in: een probleem oplossen met de hulp van een bepaald programma (werkschema) dat werkt op ingevoerde data. Dus als we de zaak goed beschouwen, kunnen we in samenhang met 'computer' net zo goed procesbesturing zeggen als rekenen (of een combinatie van beide). De uitvoer, die door de computer wordt geleverd, was vroeger een afdruk van rekenresultaten. Nu vindt er een verschuiving plaats naar *digitale outputsignalen*, (output = uitvoer), bijvoorbeeld voor het besturen van kleppen, motoren en dergelijke. In het vervolg zullen we toch het begrip computer blijven gebruiken, ook indien we het over procesbesturing hebben, ofschoon dit begrip veel te beperkt is. Als we de werkwijze van een computer nader bekijken, kunnen we de volgende stappen onderscheiden:

- de computer voert zijn bewerkingen uit aan de hand van *programma's* en *gegevens* (data). Beide zijn vooraf in het geheugen geplaatst. Dit noemen we het stored-program-computer-concept. De gegevens en de programma's worden in de computer gebracht via een inputeenheid (input = invoer) van de computer;
- de computer moet het opgeslagen programma uit kunnen voeren.


digital decwriter II
WRITE ENABLE

Hierbij worden de opdrachten van het programma in volgorde afgewerkt. Deze functie wordt verzorgd door het *rekenorgaan* (ALU = arithmetic and logical unit) en de *besturing* (control unit) welke in de computer aanwezig zijn;
- de resultaten van de bewerkingen worden daarna via een *output-orgaan* (uitvoer-orgaan) ter beschikking van de gebruiker gesteld.



Afb. 1.1. Basisblokken van een computer.

Afbeelding 1.1 laat de samenhang van de genoemde basisblokken nog eens zien. De apparatuur stelt alle hulpmiddelen voor de uitvoering van de bovengenoemde taken ter beschikking. Maar de computer wordt pas dan nuttig als er een programma in het geheugen staat, waarmee de apparatuur aan het werk kan.

1.2. Wat is programmeren?

Een computer voert bewerkingen uit aan de hand van opdrachten welke hem worden gegeven. Deze opdrachten vormen samen met de te verwerken gegevens de *input* voor de computer. De opdrachten welke de programmeur aan de machine geeft, vormen een reeks zogenaamde *statements* (statement = een opdracht voor de computer in een hogere taal) of *instructies* (instructie = een opdracht voor de computer in machinecode). Deze reeks vormt een programma.
Het maken van deze lijst met statements of instructies heet *programmeren*. Het betekent dat de programmeur moet weten welke acties de computer als gevolg van elk van zijn opdrachten zal uitvoeren. Als hij precies weet wat het effect is, kan hij de opdrachten zó gebruiken, dat de computer voor hem een aantal acties zal uitvoeren, die samen een oplossing van een door hem aangedragen probleem vormen. De op te lossen problemen zullen zó moeten worden beschreven, dat het mogelijk is dat via een reeks van de zojuist genoemde opdrachten, welke aan de computer worden gegeven, het probleem wordt opgelost. De 'kunst van het programmeren' betekent niets anders dan het geven van eenduidige opdrachten aan de computer. Deze zal dan volgens het gegeven 'recept' het resultaat produceren dat we wensen.
Een programmeur zal zich een *denkwijze* eigen moeten maken. De probleemstelling, welke door middel van een computerprogramma moet worden aangepakt, moet hij kunnen opdelen in *basisopdrachten*. Deze basisopdrachten moeten in volgorde worden uitgevoerd. Dit kan het best gebeuren door de *probleemoplossing*, die meestal veel te groot zal zijn om direct door de statements te worden gerealiseerd, op te delen in een aantal *deeloplossingen* van deelproblemen. Deze deelproblemen kunnen weer verder worden onderverdeeld. We moeten hierbij zó ver doorgaan, totdat de overgebleven delen elk door een of enkele statements kunnen worden gerealiseerd. Deze zogenaamde 'top-down' aanpak van een probleem zal meestal tot goede resultaten en overzichtelijke programma's leiden.

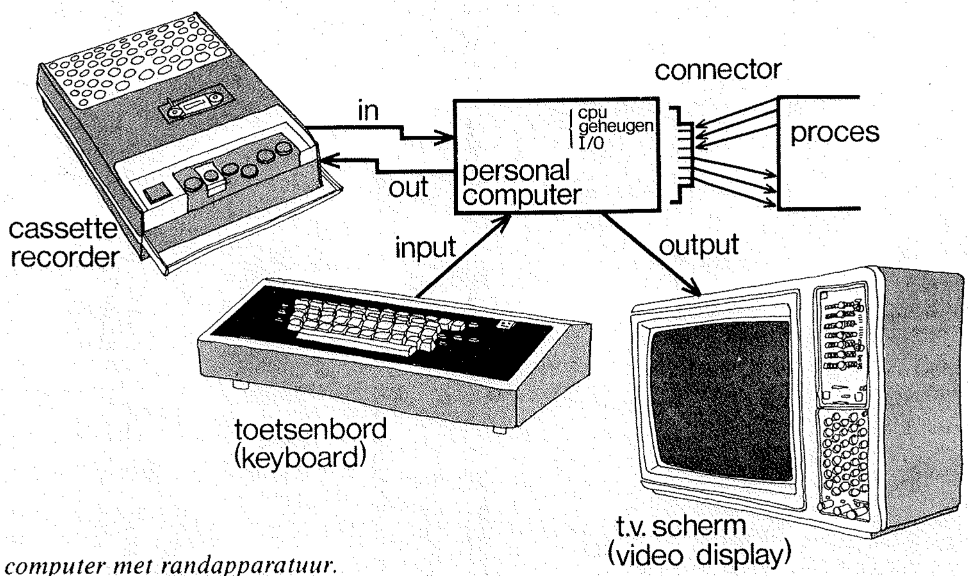
Samenvattend kunnen we dus zeggen dat programmeren betekent: het omvormen van een oplossingsmethode van een probleem tot een reeks uitvoerbare computerstatements. Deze reeks zal in volgorde worden uitgevoerd. Het resultaat van de uitvoering is dan de oplossing van het probleem.

1.3. Personal computers

Door de recente ontwikkelingen op het gebied van de micro-elektronica is een aanzienlijke miniaturisatie (verkleining) opgetreden op het gebied van de elektronische circuits. Het gevolg hiervan was een sterke prijsdaling, die samen met de kleine afmetingen heeft geleid tot een massaproduktie van microcomputers. Het resultaat van deze ontwikkeling is, dat we nu de computer van gisteren, die miljoenen kostte, voor een lage prijs en in een handelbare behuizing kunnen inzetten voor vele toepassingen. We bevinden ons nu in een periode waarin de personal (eigen) computer zijn intrede zal doen in scholen, thuis, op kantoor, enz.
De microcomputer is tot nu toe vaak benaderd vanuit de hoek van de computertechnicus of de elektronica-specialist. Steeds werd uitgegaan van het feit dat men thuis was op het gebied van elektronica, computerprogrammering of beide.
De technieken, die bij de personal computer worden toegepast, zijn echter zo eenvoudig dat ook de niet-technicus of zelfs de leek die helemaal geen technische opleiding heeft, met de huidige computer kan werken. Hij moet echter wel hebben geleerd hoe hij moet programmeren. Hierdoor zal deze zelfde (micro)computer in de nabije toekomst zijn intrede gaan doen in gebieden waaraan men tot voor kort nooit heeft durven denken.
De computer, in welke vorm dan ook, kan het leven thuis veel gemakkelijker en aangenamer maken. In principe is de computer niet moeilijker te bedienen dan een naaimachine, een wasmachine of soortgelijke apparatuur. De tendens is dat veel van deze apparaten al worden uitgerust met microcomputers zonder dat de gebruiker er iets van weet. Maar toch gebruikt hij deze machines met even veel gemak als vroeger.

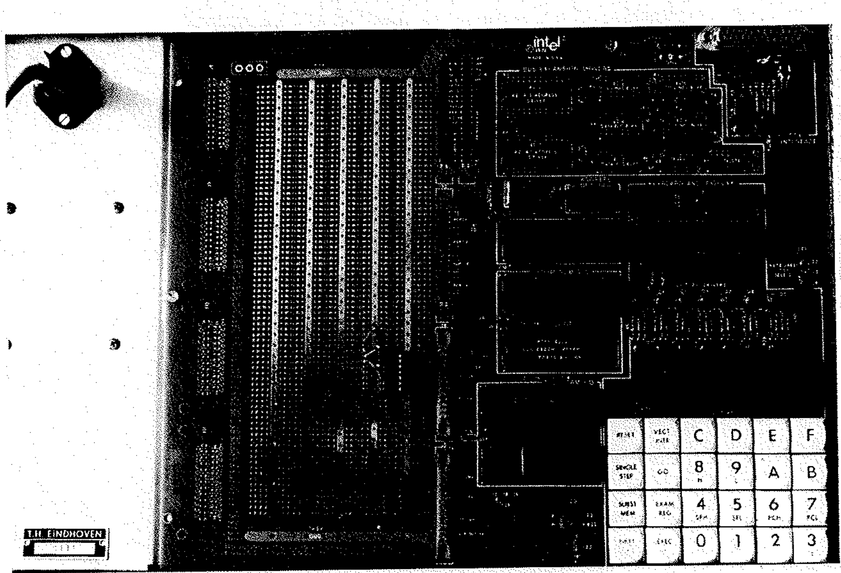
Apparatuur
Een personal computer ziet er vaak uit als een *schrijfmachine*, die is gecombineerd met een televisietoestel. Het toetsenbord (keyboard), dat in grote lijnen gelijk is aan het toetsenbord van een schrijfmachine, wordt gebruikt door de programmeur om data en programma's in te voeren. Op het *scherm* (video-display) worden de resultaten van berekeningen en dergelijke zichtbaar gemaakt. Dit zijn de belangrijkste invoer- en uitvoerapparaten, die we bij de personal computer terugvinden. Vaak zijn er nog extra mogelijkheden voor input/output om bijvoorbeeld via een connector (aansluitpen) zelf te besturen apparatuur aan te sluiten.
Als we gegevens en programma's voor een langere periode willen bewaren, kunnen we gebruik maken van normale *audio-cassetterecorders* (of meer professioneel, van een *floppy disk*). Een floppy disk is een eenvoudig schijvengeheugen.
Het zal duidelijk zijn dat zo'n systeem kan worden ingezet voor het bijhouden van het huishoudboekje, het controleren van de boek-

houding en de belastingen, maar ook voor het besturen van allerlei hobby-apparatuur. Vaak hebben we dan wel een *printer* en een *floppy disk* nodig. (Printer = afdrukapparaat).
De aanschaf van een personal computer kunnen we vergelijken met de aankoop van een stereo-installatie. We kunnen een compleet systeem kopen, maar we kunnen ook delen zelf bouwen uit onderdelen en later een veelheid van uitbreiding maken.
Afbeelding 1.2 geeft een schema van een personal computer met nog enkele mogelijke randapparaten.



Afb. 1.2. Een personal computer met randapparatuur.

Een echte computer
De computer was tot voor kort alleen een rekenmachine. Hij is nu veranderd in een machine voor van alles en nog wat. Door de lage prijs is het mogelijk geworden om één microcomputer in te zetten voor één enkele toepassing. Dit in tegenstelling tot het gebruik van een enkele computer voor vele toepassingen na elkaar. (Dit is de zogenaamde general purpose computer.) Voor een andere toepassing kan nu telkens een andere computer worden ingebouwd. Deze manier van werken komt sterk overeen met de aanpak van vroeger. Men probeerde toen om programma's zo klein mogelijk te houden. Dit in verband met de prijs van geheugens. Dit is nu helemaal achterhaald. Het is nu mogelijk om programma's zo duidelijk mogelijk te maken zonder daarbij op het geheugengebruik te letten.
De personal computer is nu een *compleet* computersysteem geworden, dat voor allerlei algemene toepassingen kan worden ingezet. Het ziet er naar uit dat de 'gewone man' door de personal computer een groot aantal wiskundige gereedschappen zal kunnen gebruiken, zonder daarbij exact de wiskundige achtergronden te kennen. We zullen de computer terugvinden in allerlei apparaten. We kunnen hierbij een vergelijking maken met het toepassen van de elektromotor, die we in veel apparaten inbouwen, zonder ons zorgen te maken over de moeilijke differentiaalvergelijkingen, die nodig zijn bij het ontwerpen van de machines.



Een microcomputer van Intel voor onderwijstoepassingen.

Realiseert een huisvrouw zich soms wat er precies gebeurt als ze een mixer gebruikt? Nee, maar toch kan ze de resultaten van het toepassen van de mixer (met de ingebouwde elektromotor) gebruiken. Zoals het met de elektromotor is gegaan, zal het in de nabije toekomst ook met de microcomputer gaan!

1.4. Programmeertalen

Doordat een computerinstallatie voor een groot aantal verschillende taken kan worden ingezet (namelijk door het verwisselen van programma's) is het zinvol eens wat dieper op deze zogenaamde *software* (programmatuur) in te gaan.
Voor het opstellen van programma's kunnen de volgende soorten programmeertalen worden toegepast, namelijk:
- *machinetalen;*
- *assembleertalen;*
- *hogere (probleemgerichte) talen.*

Machinetaal
Bij de *machinetaal* moeten de opdrachten, die door de computer moeten worden uitgevoerd, worden gecodeerd in de vorm van binaire getallen. Deze gecodeerde opdrachten kunnen door de computer direct worden uitgevoerd. Het is echter zeer moeilijk om op deze wijze programma's te schrijven, omdat:
- het programmeren op deze wijze zeer tijdrovend is;
- deze methode onoverzichtelijk is en de kans op het maken van fouten (welke dan ook weer zeer moeizaam zijn op te sporen) vergroot;
- de programma's nauwelijks leesbaar zijn.

Assembleertalen
Omdat bij het ontwikkelen van projecten, waarbij computers worden toegepast, de meeste tijd (en dus ook geld) in de software gaat zitten, is er gezocht naar een meer economische benadering van het probleem.

De *assembleertalen* geven een duidelijke verbetering in dit software-probleem te zien. In de assembleertaal programmeren we in een *symbolische* taal, dat wil zeggen dat de opdrachten, die we de computer willen laten uitvoeren, in een code worden genoteerd, die we eenvoudig kunnen lezen en begrijpen. In plaats van de binaire code voor bijvoorbeeld de opdracht optellen, gebruiken we de symbolische code 'ADD'. Dit is een notatiewijze die veel beter leesbaar is dan de machinecode. Er moet nu wel een extra (geautomatiseerde) stap worden gemaakt.
De programma's, die zijn geschreven in een symbolische taal, moeten namelijk nog worden omgezet in *binaire* patronen. Want alleen deze binaire patronen kunnen door de computer worden uitgevoerd. Bij programma's, die in een assembleertaal zijn geschreven, moeten alle door de computer uit te voeren stappen in *symbolische code* worden genoteerd. Een nadeel is dat de taal geheel op een specifieke machine is gericht. Gewoonlijk is deze taal van het Engels afgeleid, zoals 'ADD' voor optellen al aangaf. Dit houdt in dat de uitwisseling van programma's tussen verschillende typen computers nauwelijks mogelijk is. Een voordeel is echter wel dat de programma's, die in assembler worden geschreven, snel, kort en daardoor dus efficiënt kunnen zijn tijdens de *executiefase* (= de periode dat het programma wordt uitgevoerd). De programmeerfase is echter complex en langdurig. Bovendien wordt hierbij veel kennis van de gebruiker gevraagd wat betreft de structuur van de machine en de *instructieset* (instructieset = verzameling van alle machine-instructies).
Voor sommige (bijvoorbeeld tijdkritische of sterk op input of output gerichte) programma's kan de assembleertaal echter de beste oplossing zijn.

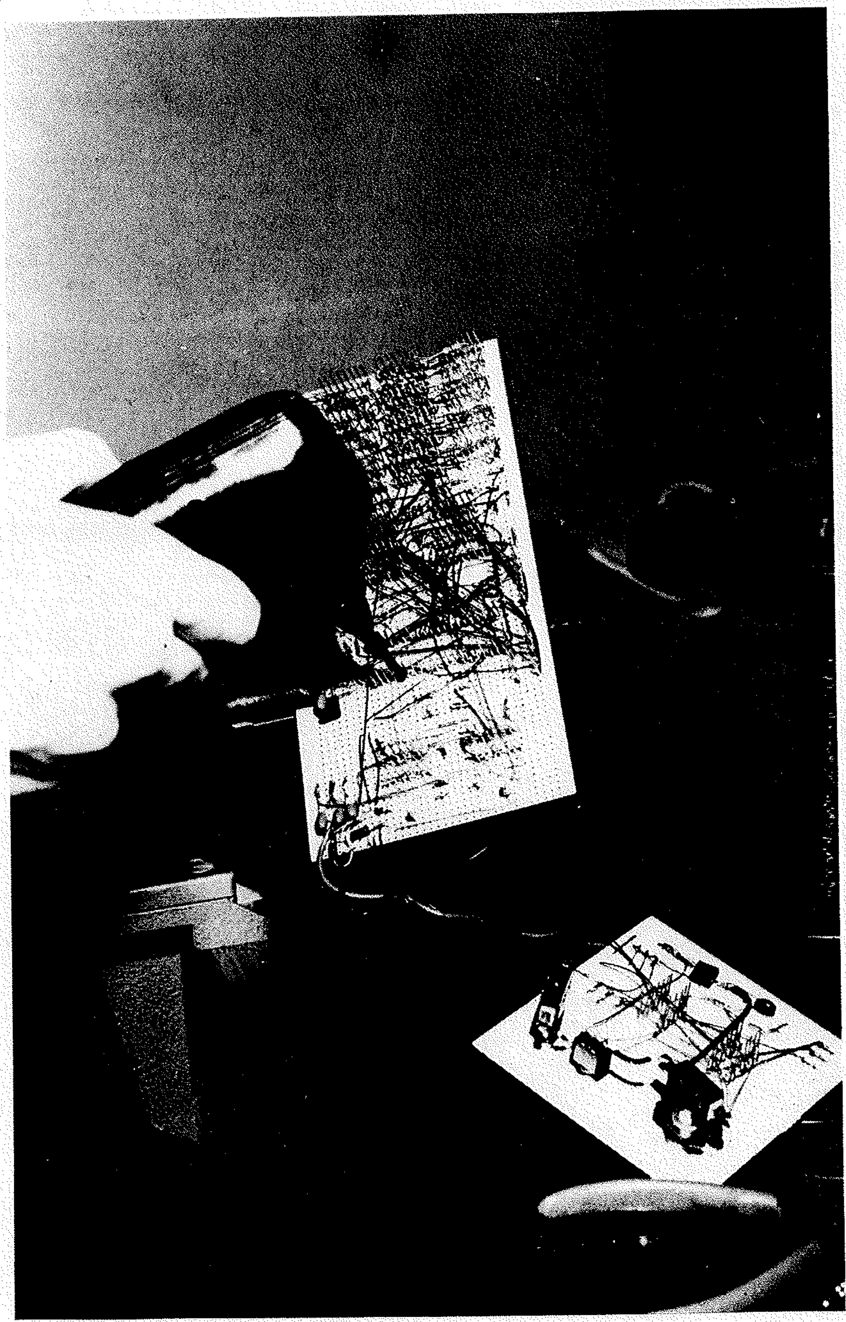
Hogere programmeertalen
Bij hogere programmeertalen vallen veel van de genoemde nadelen weg. De hogere talen zijn *probleem-georiënteerd* en *niet machine-afhankelijk*. Ze zijn in het algemeen gebaseerd op wiskundige formules en bestaan uit opdrachten welke worden genoteerd in een taal, die sterk doet denken aan het Engels.
De hogere talen hebben over het algemeen de volgende voordelen:
- ze zijn overzichtelijk;
- het programmeren kost weinig tijd in verhouding tot het programmeren in machinetaal;
- ze zijn eenvoudig te leren;
- ze zijn niet machine-gebonden;
- ze zijn zelf-documenterend.

Enkele bekende hogere programmeertalen zijn:

ALGOL	APL	BASIC
COBOL	FORTRAN	PASCAL
PL/I	PLM	

Beginners All-purpose Symbolic Instruction Code
Van de bovengenoemde talen is BASIC eenvoudig, gemakkelijk te leren en goedkoop te implementeren (in een computer in te bouwen). Het is daardoor een geschikte taal voor beginners. Toch bevat BASIC dezelfde benaderingsmethodieken welke in andere hogere program-

meertalen voorkomen. Door BASIC te leren, maakt men kennis met de algemeen toegepaste denkwijze, waardoor het later gemakkelijk wordt om over te stappen op andere talen.
Een ander voordeel van BASIC is het *interactieve* karakter van deze taal. Dit betekent dat het programmeren en het uitvoeren van programma's in de vorm van een dialoog tussen de computer en de gebruiker verloopt. Ze zullen om de beurt de besturing in handen krijgen, waardoor het mogelijk wordt stap voor stap het programma op te bouwen en te testen. BASIC geeft tijdens de programmavoorbereiding en de executie duidelijke foutmeldingen en geeft direct het resultaat. Men hoeft niet op zijn beurt te wachten zoals vaak het



Bouw van een microcomputer. Het prototype wordt gemaakt door middel van de zogenaamde wire wrap techniek. Hierbij worden dunne draden gedraaid om de pennen van de onderdelen.

geval is in computercentra. Bij de programmeertaal BASIC hebben we de computer 'helemaal' voor onszelf. Dit geldt voor de personal computer, maar ook voor verbindingen met grote computercentra via *time-sharing*. (Time-sharing = het werken met een computer door vele gebruikers tegelijk. Ieder heeft de machine dan een korte tijd ter beschikking.) Daarbij merken we niet of nauwelijks dat er nog andere gebruikers zijn.

De programmeertaal BASIC is in 1963 door dr John G. Kemeny en dr Thomas E. Kurtz aan het *Dartmouth College* ontwikkeld. Het ontwikkelen van de taal was een project dat werd gefinancierd door de Amerikaanse *National Science Foundation*.
BASIC is een afkorting van:

*B*eginners *A*ll-purpose *S*ymbolic *I*nstruction *C*ode.

Dit betekent (vrij vertaald): een symbolische taal voor beginners die bestaat uit algemeen toepasbare opdrachtencodes.
BASIC is op bijna alle computersystemen beschikbaar (ook op de microcomputersystemen). Door de jaren heen zijn er vele varianten van de BASIC van 1963 ontstaan, maar de kern van al deze 'dialecten' is gelijk. BASIC vinden we terug in de meeste personal computers.
Oorspronkelijk is BASIC opgezet voor technici, wetenschappers, statistici, zakenlieden, leraren en studenten. Tegenwoordig is BASIC de 'people's computer language' (*volkscomputertaal*), omdat nu ook kinderen op lagere scholen en hobbyisten deze taal gebruiken.

Voordelen van BASIC:
- BASIC wordt veel gebruikt;
- BASIC is goedkoop;
- BASIC is eenvoudig te leren.

Nadelen van BASIC:
- sommige versies van BASIC zijn slecht gestructureerd;
- BASIC is traag in de verwerkingssnelheid.

1.5. Programmeervoorbereiding

Als we een computerprogramma willen maken, moeten we bij de ontwikkeling ervan twee belangrijke stappen onderscheiden, namelijk:
- de *probleemdefinitie-fase*, dit wil zeggen een fase waarin het op te lossen probleem of opgave exact wordt geformuleerd;
- de *analyse-fase*, dit is de fase waarin de opgave en de onderdelen waarin deze is op te splitsen, zo worden 'vertaald' dat de deelproblemen door middel van programma's zijn op te lossen.

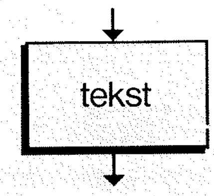
Verbale beschrijving
In werkelijkheid betekent dit dat het probleem eerst in 'gewone' spreektaal moet worden beschreven (*verbale beschrijving*). Dit geldt ook voor de resultaten welke we willen verkrijgen bij de probleemoplossing.
Hierna moeten we deze 'verbale beschrijving' verder bestuderen om

een uitspraak te kunnen doen of het probleem wel per computer is op te lossen. Indien dit blijkt te kunnen, moeten we de methode van oplossen verder uitwerken. Tevens is het nuttig om na te gaan of er *alternatieve* oplossingsmethoden zijn. Zo zal er een redelijke oplossing ontstaan, die goed past bij het gestelde probleem. Bij de keuze uit de alternatieven spelen aspecten zoals snelheid, grootte van het programma en dergelijke een rol.

Vertalen en testen
De volgende stap is het *vertalen* van de uitgewerkte oplossingsmethode in een voor de computer geschikte programmeertaal. De laatste fase (welke niet bij de programmavoorbereiding hoort) is het *testen* van het geschreven programma op de computer en wel voor alle mogelijke condities.
Deze condities moeten we, als het mogelijk is, zoveel mogelijk al tijdens de probleem-definitiefase hebben voorzien en beschreven. Wanneer dit wordt vergeten, is het optreden van fouten in programma's heel vaak het gevolg. Bovendien moeten de programma's ook nog worden voorzien van *commentaar* en worden *vastgelegd* in een of andere vorm van *documentatie*.

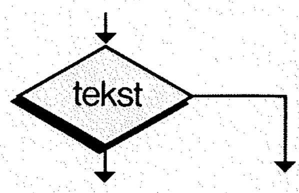
1.6. Stroomschema's (flow-diagrammen)

Nadat de programmavoorbereiding een gunstige oplossingsmethode heeft opgeleverd, is het vaak zinvol om van het programma een *stroomschema* te maken. Hierin worden door middel van grafische symbolen de uit te voeren handelingen vastgelegd samen met de volgorde waarin de handelingen moeten worden uitgevoerd.



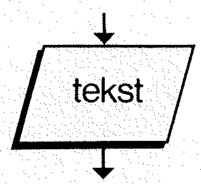
Afb. 1.3. Symbool voor een of meer algemene operaties.

Symbolen
De belangrijkste symbolen die vaak bij stroomschema's worden gebruikt, vinden we in de afbeeldingen 1.3 tot en met 1.11.
Het meest gebruikte symbool is de *rechthoek*. Dit symbool geeft een reeks bij elkaar horende handelingen of een of meer standaardoperaties aan. De tekst in het blok geeft de gebruiker een duidelijke indicatie wat er zoal in zo'n blok wordt uitgevoerd. Door middel van de ingaande en uitgaande pijlen wordt weergegeven waar we een blok binnenkomen en waar we naar toe gaan als we het blok verlaten. Afbeelding 1.3 laat dit blok zien.



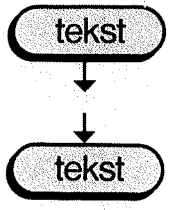
Afb. 1.4. Symbool voor een vertakking.

Een tweede en vaak toegepast symbool is in afbeelding 1.4 gegeven. Dit symbool stelt een *vertakking* in een programma voor. We komen op een plaats in het blok binnen, er moet dan een *beslissing* (decision) worden genomen op welke van de twee manieren het blok zal worden verlaten. Welk pad we bij het verlaten zullen nemen, hangt af van de uitspraak die er in het blok wordt gedaan. De tekst van het vertakkingsblok bevat namelijk een uitspraak. Als de uitspraak waar is, wordt het blok verlaten op de plaats waar T (true; ja; waar) staat. Wanneer de uitspraak niet waar is, wordt de vervolgrichting met de tekst F (false, nee of niet waar) genomen.



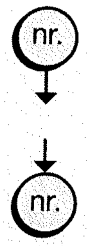
Afb. 1.5. Symbool voor invoer of uitvoer.

Afbeelding 1.5 geeft een symbool van een blok waarin op een andere manier *data-uitwisseling* met de buitenwereld geschiedt. De tekst in het blok zal aangeven of er sprake is van input of output. Vaak wordt aangegeven welke variabele hierbij wordt gebruikt.
In afbeelding 1.6 wordt het *beginpunt* en het *eindpunt* van een



Afb. 1.6. Begin- en eindpunt van een programma.

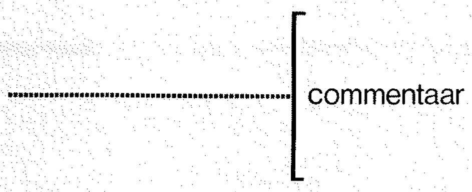
programma of programmadeel weergegeven. De tekst geeft de naam van het programma aan.
Vaak worden grotere programma's onderverdeeld in stukken. Dit gebeurt om de overzichtelijkheid te bevorderen. Om de samenhang tussen de diverse onderdelen te laten zien, worden *overgangspunten* (connectoren) gebruikt met daarin een getal of een naam. De pijl bij deze blokken geeft aan waar we naar toe gaan of waar we vandaan komen. Zie hiervoor afbeelding 1.7.
Om extra *commentaar* toe te kunnen voegen gebruiken we de ruimte naast het stroomschema. Door middel van een *stippellijn* en een *vierkante haak* geven we aan op welk blok het commentaar betrekking heeft. Zie afbeelding 1.8.
Al enkele malen zijn de *pijlen* ter sprake gekomen. Door de pijl wordt namelijk in het stroomschema de volgorde van handelen aangegeven. Naast de bovengenoemde symbolen komen nog vaak de symbolen van afbeeldingen 1.9, 1.10 en 1.11 voor. Ze worden toegepast voor het aanroepen van *subroutines,* voor het uitvoeren van *meervoudige vertakkingen* en het vormen van *programmalussen*. Een en ander komt in de volgende hoofdstukken nader aan de orde.



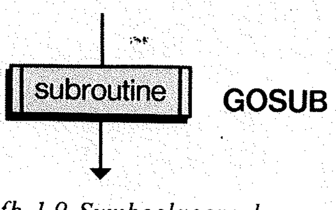
Afb. 1.7. Begin- en eindpunt van deelprogramma's of knooppunt in een programma.

1.7. Voordelen van het gebruik van stroomschema's

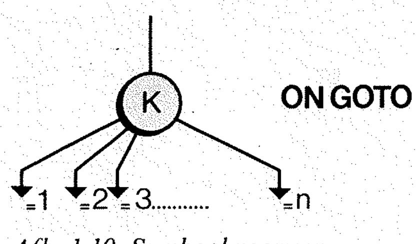
Een stroomschema blijkt in praktijk erg nuttig te zijn bij het *ontwikkelen* van programma's. De volgende voordelen komen hierbij naar voren:
- programma's worden bij gebruik van de stroomschema's overzichtelijk en gestructureerd geschreven;
- een stroomschema geeft de programmeur een duidelijk inzicht in de logische opbouw van een programma;
- door het stroomschema moet de programmeur alle mogelijke richtingen in een programma volledig uitwerken (eventueel doodlopende richtingen of vergeten paden komen tijdig naar voren);
- het stroomschema kan als een beschrijving van het programma dienen en als een stuk documentatie worden beschouwd. Het kan zo als discussiestuk tussen twee gebruikers dienen;
- het zoeken naar fouten kan door het gebruik van de stroomschema's sterk worden vereenvoudigd.



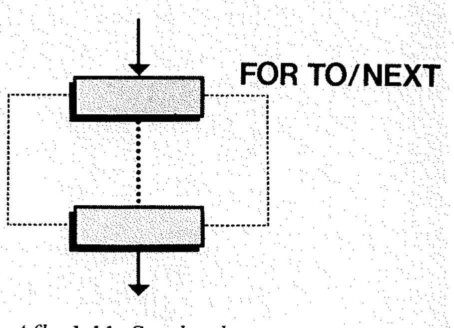
Afb. 1.8. Commentaar bij programma's.



Afb. 1.9. Symbool voor subroutines.



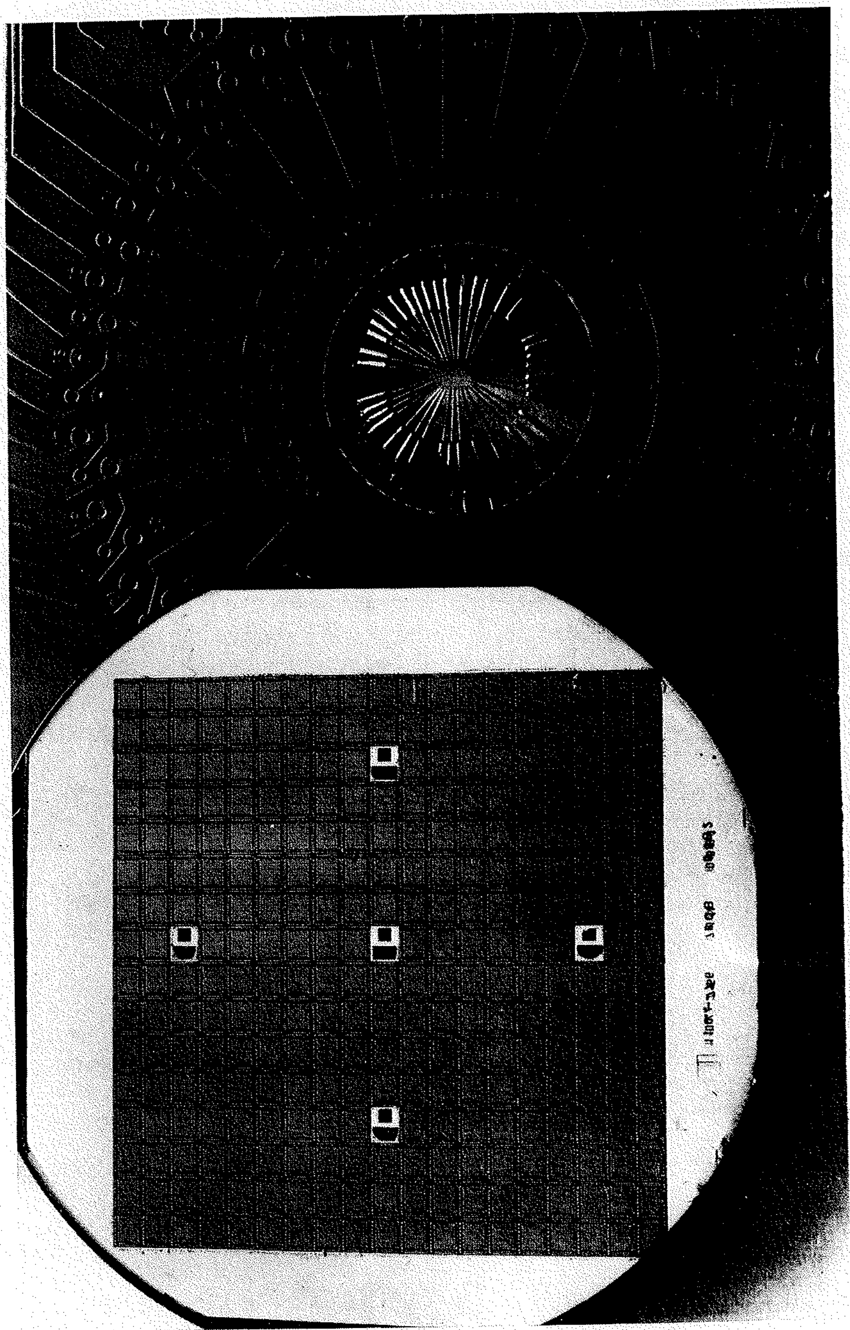
Afb. 1.10. Symbool voor een meervoudige vertakking.



Afb. 1.11. Symbool voor een programmalus.

Samenvatting hoofdstuk 1

In dit hoofdstuk is globaal verteld wat programmeren is. Bovendien is verteld dat via stroomschema's een probleem eenvoudiger kan worden opgelost. De programma's kunnen dan via deze stroomschema's worden gerealiseerd.



Via een bundel van meetpennen wordt de microcomputer getest op correcte werking.

Hoofdstuk 2

Basisbegrippen

In dit hoofdstuk maken we een eenvoudig BASIC programma. Dit zal zonder veel voorkennis duidelijk zijn. Hiermee willen we aantonen dat programmeren helemaal niet moeilijk is.
Bovendien zullen een aantal elementaire begrippen aan de orde komen, die later vaak zullen terugkomen.

2.1. Een eenvoudig voorbeeld in BASIC

Ter introductie kiezen we een eenvoudig *voorbeeld* van een programma, geschreven in de programmeertaal BASIC. Het rekenkundige deel van het programma is zo simpel dat iedereen het zonder meer moet kunnen begrijpen.
Het probleem, dat moet worden opgelost, is het bepalen van het produkt van twee getallen. Deze getallen worden via het toetsenbord van de *systeemterminal* ingevoerd. (Systeemterminal = in geval van een personal computer een combinatie van het *toetsenbord* en het *televisiescherm*. Bij andere systemen is het ook soms een *teletype*. Dit is een combinatie van een *printer, toetsenbord, ponsbandlezer* en *ponsbandponser.*) Het resultaat moet op de systeemterminal zichtbaar worden gemaakt of afgedrukt. In plaats van systeemterminal wordt ook wel de naam terminal gebruikt.
Afbeelding 2.1 laat eerst het programma zien. Achter de afdruk van het programma zien we dat het programma wordt gestart en wat er gebeurt tijdens de executie ervan.

```
*LIST
10    INPUT A
20    INPUT B
30    LET P=A*B
40    PRINT A,B,P
50    END
*RUN
?10
?17
 10            17           170
END PRØGRAM
*RUN
?1.5
?2.4
 1.5           2.4          3.6
END PRØGRAM
```

Afb. 2.1. Programma in BASIC dat tweemaal wordt uitgevoerd.

Nadat het programma in het geheugen van de computer is gezet (hoe dit gebeurt komt later aan de orde), wordt door middel van het commando RUN, gevolgd door het aanslaan van de RETURNtoets, het programma ten uitvoer gebracht (gestart).
Via de RETURNtoets geven we aan de computer te kennen, dat we klaar zijn met het invoeren van een *statement* of een *commando* en dat de computer weer verder mag gaan met het programma en niet langer op invoer vanuit het toetsenbord hoeft te wachten. Hij zal dan ook zelf het initiatief nemen en doorgaan met het programma. Als er geen programma meer uit te voeren is, vraagt de computer aan de gebruiker om een nieuw commando.
De computer moet in het voorbeeld van twee in te voeren getallen het produkt bepalen. Tijdens de executie van het programma wordt door de computer via het afdrukken van een ?-teken op de systeemterminal om de *getalwaarde* van het eerste getal gevraagd. De gebruiker van het programma moet nu het initiatief nemen en het getal (A) intikken op de systeemterminal. In het geval van afbeelding 2.1 wordt het getal 10 en daarna RETURN ingetoetst. De computer kan nu verder gaan met het programma en via een ?-teken om het tweede getal (B) vragen. Na het invoeren van de betreffende getalwaarde *berekent* de computer het produkt P en voert de waarden van A, B en het produkt uit. Dan geeft de computer door middel van de boodschap. END PROGRAM, gevolgd door een stervormig teken (*) op de daarop volgende regel aan, dat de gebruiker kan kiezen uit bij voorbeeld een nieuw commando, het invoeren van een nieuw programma of het nog eens tot uitvoering brengen van het programma. Zoals in het voorbeeld met andere waarden voor A en B.

Nu zullen we ingaan op het programma zelf. Elke (genummerde) regel van het programma bevat een BASIC statement, dat door de computer kan worden uitgevoerd. Hierbij worden de namen *variabele* en *expressie* gebruikt. Deze zullen verderop in dit boek worden uitgelegd.
In regel 10 wordt door middel van het INPUT statement aan de computer de opdracht gegeven om *data* in te lezen. De data zal op het toetsenbord van de systeemterminal moeten worden ingetoetst.
De gebruiker van het programma krijgt tijdens de executie van het INPUT statement door een ?-teken de mededeling dat de computer staat te wachten op de *invoer* van de data. Nadat het getal door de gebruiker is ingetoetst (gevolgd door RETURN) krijgt de in het betreffende INPUT statement genoemde variabele A de waarde van het getal en de computer is nu helemaal klaar met het statement en kan verder gaan met het volgende statement. Dit is het statement, dat op de regel staat met het regelnummer dat volgt op de regel, die zojuist is uitgevoerd.
(Als er gaten in de nummering zijn, wordt het nummer genomen dat onmiddellijk volgt op het zojuist genoemde nummer.) In ons geval is dit de regel met het nummer 20.
De nummering van de regels mag in elke oplopende volgorde worden gekozen.
Het is echter wel handig om hierbij stappen van tien te maken. Er kan dan later nog gemakkelijk een aantal regels worden tussengevoegd.

In regel 20 krijgt de variabele B de waarde van het getal dat nu door de gebruiker wordt ingetikt.
Nu volgt regel 30. Bij de uitvoering van het LET statement wordt de waarde van de *expressie* A * B (expressie = een of andere berekening) bepaald en door middel van *toewijzing* (=) gegeven aan de variabele P. In regel 40 krijgt de machine door middel van het PRINT statement de opdracht om de waarden van de variabelen A, B en P achtereenvolgens op de *terminal* zichtbaar te maken.
In de volgende regel (50) staat de END opdracht. Dit is geen opdracht waarbij de computer een actie moet uitvoeren, maar het is een woord waaraan de computer herkent dat het programma nu is afgelopen. De computer zal zich nu melden door op de terminal de tekst END PROGRAM gevolgd door een *-teken (op de volgende regel) zichtbaar te maken. Nu mag de gebruiker bijvoorbeeld een nieuw commando geven.
In het algemeen mogen we zeggen dat een statement een *opdracht* is, die tijdens de executie van het programma wordt uitgevoerd. Een commando wordt echter direct ten uitvoer gebracht na het intikken ervan. Vaak wordt deze scheiding echter niet consequent doorgevoerd.

Opmerking: Er bestaan vele soorten BASIC. Dat wil zeggen dat er veel verschillen voorkomen wat betreft de commando's, statements en de foutmeldingen.

2.2. Het invoeren van een programma in de computer

Wanneer we een programma op papier hebben voorbereid (dit wil zeggen door middel van een *voorstudie, flowdiagram* en *lijst* met de uit te voeren statements) dan willen we het programma invoeren in de computer. Om dit mogelijk te maken moeten we het *geheugen* eerst vrijgeven. Dit wil zeggen, dat indien er nog een programma in het geheugen zou staan, dit programma eerst moet worden verwijderd. Dit kunnen we doen door op de systeemterminal het commando NEW in te tikken. Zie afbeelding 2.2.

```
*NEW
*10 INPUT A
*20 INPUT B
*30 LET P=A*B
*40 PRINT A,B,P
*50 END
*LIST
10    INPUT A
20    INPUT B
30    LET P=A*B
40    PRINT A,B,P
50    END
*
```

Afb. 2.2. Invoeren van een programma.

De computer is nu klaar voor de invoer van een nieuw programma. Dit programma wordt regel voor regel ingetikt, waarbij we zowel het *regelnummer* als het *statement* intikken. Na het invoeren van het programma kunnen we zien of de tekst correct in het geheugen staat door het programma (dat wil zeggen de *programmatekst*) naar de systeemterminal te voeren. Hiervoor is het commando LIST beschikbaar. We gaan er nu nog van uit dat het hele programma correct is ingetikt.
Het verbeteren van fouten komt later nog aan de orde (hoofdstuk 3).
Na het intikken van het programma zal nu de computer klaar zijn om dit programma ten uitvoer te brengen.
In het voorgaande programma hebben we de volgende BASIC *sleutelwoorden* (keywords) leren kennen.

Statements:	*Commando's:*
INPUT	NEW
LET	LIST
PRINT	RUN
END	

Om een overzicht te krijgen van de mogelijkheden van BASIC zullen we in volgende hoofdstukken de diverse statements in de vorm van enkele voorbeelden toelichten en daarna een formele beschrijving van de besproken statements geven. Een formele beschrijving geeft precies aan wat het statement betekent.

2.3. Basisbegrippen voor hogere talen

Voordat we verder zullen ingaan op de verschillende mogelijkheden van hogere programmeertalen, moeten we eerst een aantal *basisbegrippen* behandelen. Dit zal een taai deel worden, maar door nu de definitie van een aantal begrippen verder uit te leggen, zullen we ons later helemaal kunnen beperken tot de dan te bespreken stof. We richten ons hierbij vooral op begrippen die in BASIC voorkomen.

Statements
Een BASIC programma bestaat uit regels, die elk een BASIC statement bevatten.
Elke regel begint met een regelnummer. Dit nummer dient ter identificatie van de regel. De computer zal de regels in *volgorde* van de regelnummers uitvoeren.
Bovendien kunnen de regelnummers in programma's worden gebruikt voor het aangeven van de plaats waar we bij *sprongen* naar toe willen springen. Het statement dat in een regel voorkomt is te beschouwen als een *opdracht* die de computer ten uitvoer moet brengen.
Het statement begint met een sleutelwoord (keyword) dat aangeeft wat er door de computer moet worden gedaan. BASIC kent een aantal van deze keywords zoals bij voorbeeld PRINT, LET, READ, enz. Ze komen later aan de orde. Vaak zal samen met de opdracht die moet worden uitgevoerd, de bijbehorende data of een verwijzing hiernaar in het statement zijn opgenomen. Over het algemeen zal een statement niet langer zijn dan één regel. Ook zal er op één regel niet meer dan één statement mogen voorkomen. (Bij sommige soorten computers komen hierop wel eens uitzonderingen voor, zo ook bij de voorbeelden uit dit boek.)
In een statement mogen vrij *spaties* (blanco's) worden tussengevoegd. Hierdoor kan vaak de leesbaarheid worden opgevoerd. Dit heeft echter geen invloed op de uitvoering van het programma. Wel moeten we opletten dat er geen spaties voorkomen in keywords of namen van variabelen, want dan gaat het wel fout. De computer herkent de namen niet meer.

Constanten
Een *constante* kunnen we vergelijken met een *getal.* Een constante heeft een vaste waarde. Deze waarde verandert nooit. Een constante is dus een onveranderlijke grootheid. De constanten zijn in twee groepen onder te verdelen, namelijk in:
- *numerieke constanten* (ofwel getallen);
- *alfanumerieke constanten* (ofwel karakters = letters, cijfers en leestekens).

De getallen vinden we in de praktijk in de vorm van *gehele* en *gebroken getallen.*

Gehele getallen
De gehele getallen (. . ., −3, −2, − 1,0,1,2,3, . . .), die in de computertechniek ook wel *integers* worden genoemd, worden in BASIC als volgt genoteerd:

+ / − CIJFERREEKS

Deze *notatiewijze* betekent dat we een reeks cijfers hebben welke vooraf mogen worden gegaan door een teken. Hierbij wil − zeggen *minus* en + wil zeggen *plus*. Zoals ook in de wiskunde gebruikelijk is mag ook in BASIC bij positieve getallen het +-teken worden weggelaten.

Gehele getallen

Wiskundige notatie	BASIC notatie
7	7
+ 11 of 11	11
−512	−512

Gebroken getallen
De *gebroken* getallen (in de computertechniek *reals* genoemd) zijn op twee manieren te schrijven, namelijk als:

+ / − CIJFERREEKS . CIJFERREEKS
+ / − CIJFERREEKS . CIJFERREEKS E + / − exponent

(We gebruiken hiervoor respectievelijk de namen *fixed point* (vaste komma) en *floating point* (drijvende komma) notatie.)
N.B. Als de cijferreeks voor de decimale punt de waarde Ø heeft, dan mag deze worden weggelaten.

Over de bovenstaande notatiemethode kunnen de volgende opmerkingen worden gemaakt. Bij de eerste methode hebben we een gebroken getal dat bestaat uit twee delen, die van elkaar worden gescheiden door een decimale punt. In de normale schrijfwijze gebruiken we de komma, maar in de Engelstalige landen neemt men de punt. Deze punt wordt ook in de computertechniek algemeen toegepast voor het scheiden van het deel *vóór* en *achter* de 'komma'. Ook hier wordt het + en − teken gebruikt voor positieve en negatieve getallen.

Gebroken getallen

Wiskundige notatie	BASIC notatie	Teken	Gehele deel	Fractie
7,0	7.0	+	7	.0
+9,1	9.1	+	9	.1
− 191,07	− 191.07	−	191	.07

(Fractie is het deel achter de .)

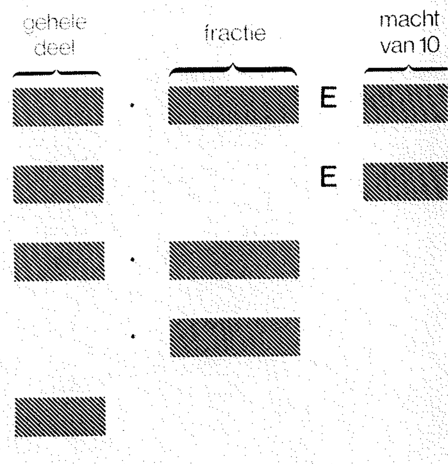
De tweede notatiemethode bevat als extra nog de letter E gevolgd door de exponent. Deze notatie komt overeen met de wetenschap-

pelijke schrijfwijze van getallen. De combinatie van de letter E gevolgd door een positieve of negatieve exponent betekent dat het voorgaande getal moet worden *vermenigvuldigd* met het getal 10 tot de macht die wordt aangegeven door de exponent.

Gebroken getallen

Wiskundige notatie	Getal-waarde	BASIC notatie	Teken	Gehele deel	Fractie	Macht van 10
$5{,}07.10^{3}$	5070	5.07E3	+	5	.07	3
$6{,}12.10^{-2}$	0,0612	6.12E−2	+	6	.12	−2
$-0{,}04.10^{6}$	−40000	−4.00E4	−	4	.00	4
6000.10^{-7}	0,0006	6.00E−4	+	6	.00	−4

In de computer waarop een BASIC programma moet worden uitgevoerd zijn vaak een aantal beperkingen van kracht wat de lengte van deze cijferreeksen betreft. Raadpleeg hiervoor de handleiding van uw computer. In afbeelding 2.3 vinden we een overzicht van een aantal verschillende getalvoorstellingen.



Afb. 2.3. Een aantal vormen van getallen in BASIC.

Strings

Naast de *numerieke* constanten bevat de programmeertaal BASIC ook nog *alfanumerieke* constanten (vaak *strings* genoemd). Deze alfanumerieke constanten bestaan eenvouding uit een reeks van letters, cijfer of leestekens. Deze tekens worden samen de *characters* (karakters) genoemd. De strings worden in de programma's toegepast om hiermee teksten te kunnen verwerken. De strings moeten in de programma's altijd worden ingesloten door aanhalingstekens (").

String	Alfanumerieke waarde
"abcd"	abcd
"EEN"	EEN
"Nummer 1"	Nummer 1
"A16B"	A16B

Variabelen

In de wiskunde worden in formules vaak *variabelen* (zoals x, y en z) gebruikt. Dit zijn namen waarvoor een getal kan worden ingevuld, waarna de betreffende formule kan worden opgelost. Ook in de hogere programmeertalen komen de variabelen voor op een wijze die overeenkomt met de normale wiskunde.

Voorbeeld:

$$X = Y + 16$$

De variabele X krijgt de waarde van de variabele Y met daarbij opgeteld de waarde 16.

Naam van een variabele
Een variabele in BASIC kunnen we *bewerken* (aanroepen, opslaan, een waarde toekennen). Hierbij gebruiken we de naam van de variabele om de waarde aan te wijzen (te *identificeren*). In het Engels heet een variabele dan ook *identifier*.

In BASIC bestaat de naam van een variabele uit een letter, een letter gevolgd door een cijfer, of een letter gevolgd door een dollarteken ($). (Soms mogen voor één naam meer letters worden gebruikt.)
De variabele naam is gekoppeld met de *data*, die bij de variabele hoort. De naam is te beschouwen als het *symbolische* adres dat aangeeft op welke plaats de bij de variabele behorende data in het geheugen van de computer staat.

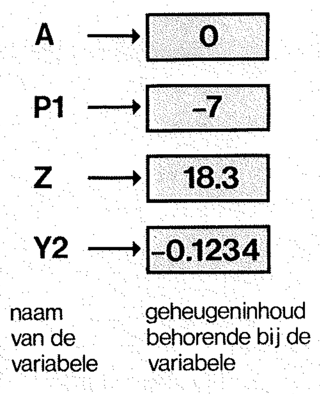
Er zijn in BASIC *twee* soorten variabelen, namelijk:
- *enkelvoudige (scalaire) variabelen;*
- *meervoudige (array) variabelen.*

Op de tweede soort variabelen wordt in een later hoofdstuk verder ingegaan, hier zullen we alleen *enkelvoudige* variabelen bespreken.

Numerieke variabelen
Een *numerieke* variabele is een variabele, die wordt gebruikt om *getallen* aan te duiden. De naam van een enkelvoudige numerieke variabele bestaat uit een letter of uit een letter gevolgd door een cijfer.
Namen van numerieke variabelen zijn bijvoorbeeld:

A	P1	W!
Z	Y2	X%

Aan een *enkelvoudige* numerieke variabele kunnen we een getalwaarde toekennen. De *initiële* waarde (de waarde bij het begin van een programma) wordt bij sommige computers gelijk aan 0 gesteld. Zo kunnen de bovengenoemde variabelen gekoppeld zijn aan de *geheugenlocaties* waarin een getalwaarde staat zoals in afbeelding 2.4. wordt aangegeven.
Sommige computers hebben de mogelijkheid om een variabele te *definiëren*, die altijd als een geheel getal zal worden afgebroken. Hiertoe voegen we dan achter de naam van de variabele een procentteken (%) toe.



Afb. 2.4. Verband tussen de naam van een variabele en de geheugeninhoud.

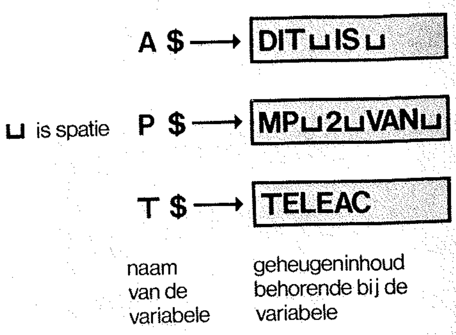
Opmerking: De Teleac/DAI computer heeft de mogelijkheid om in de naam van de variabele aan te geven of het een *real-*, *integer-* of *stringvariabele* is. Dit gebeurt door achter de naam respectievelijk een uitroepteken (!), een procentteken (%) of een dollarteken ($) te zetten. Is zo'n teken niet toegevoegd dan wordt een realvariabele verondersteld.

Alfanumerieke variabelen
Een *alfanumerieke* variabele wordt niet gebruikt om een getalwaarde aan te duiden. Hij wordt toegepast om letter-cijfercombinaties (*characters*) voor te stellen. Globaal kunnen we zeggen dat met dit soort variabelen *tekst* kan worden verwerkt.
De naam van een alfanumerieke variabele bestaat uit een letter, die

wordt gevolgd door een dollarteken ($).
Namen van alfanumerieke variabelen zijn bijvoorbeeld:

A$
P$
T$

Het aantal *characters* dat aan een BASIC variabele kan worden toegekend hangt af van het type computer. Initieel bestaat de waarde van een alfanumerieke variabele uit spaties (blanco's). Zie afbeelding 2.5.



Afb. 2.5. Alfanumerieke waarden in het geheugen.

Operatoren
In BASIC kunnen een aantal operaties op variabelen en constanten worden uitgevoerd. Deze operaties komen sterk overeen met de operaties die de wiskunde kent. Enkele algemeen bekende *basisoperaties* zijn optellen, aftrekken, vermenigvuldigen, delen enz.
Een operatie voert een handeling uit die door de *operator* wordt aangegeven. De waarden waarop de handelingen worden uitgevoerd, heten *operanden*.
Optellen van de waarde van variabele A en de waarde van variabele B:

A + B	expressie (rekenkundige bewerking)
A en B	operanden (namen van variabelen)
+	operator (optellen)

In BASIC komen de volgende soorten operatoren voor:
- *rekenkundige operatoren;*
- *relationele operatoren;*
- *logische operatoren;*
- *string operatoren.*

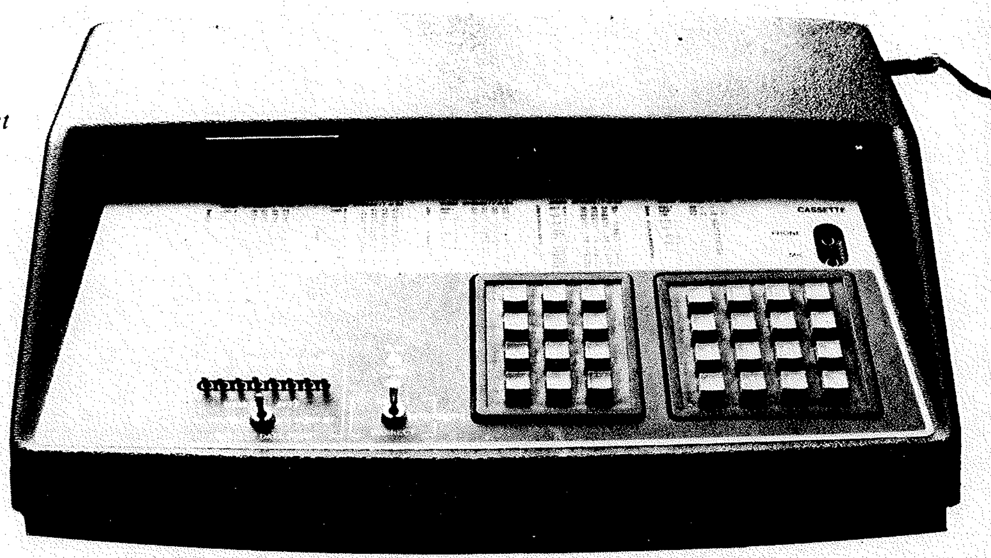
De laatste drie soorten operatoren komen later nog aan de orde.

Rekenkundige operatoren
Rekenkundige operatoren werken op numerieke operanden en geven als resultaat van de bewerking een *numerieke* waarde. Dit wil zeggen dat de operanden numerieke *variabelen* of numerieke *constanten* moeten zijn. De rekenkundige operatoren werken op de getallen (en dus niet op letters). Het resultaat van zo'n bewerking is ook weer een getal.
De rekenkundige operatoren in BASIC zijn:

Operator	Operatie	Voorbeeld	
+	optellen	A + B	tel de waarde van A en B bij elkaar op
–	aftrekken	P – Q	trek de waarde Q af van die van P
*	vermenigvuldigen	Z1*Y	vermenigvuldig de waarde van Z1 met de waarde van Y
/	delen	X/P	deel de waarde van X door de waarde van P
↑	machtsverheffen	X↑A2	verhef de waarde van X tot de macht A2

Opmerking: Voor vermenigvuldigen wordt het ∗-teken gebruikt en voor het machtsverheffen het ↑-teken. Dit in tegenstelling tot de wiskunde. Daar wordt voor vermenigvuldigen meestal het ×-teken of de punt gebruikt. Machtsverheffen wordt daar aangegeven door de macht een halve regel hoger te zetten. In BASIC gebruikt men de operator ↑ (of ∧) voor machtsverheffen.

Een Philips/Signetics microcomputer welke wordt gebruikt bij het bestuderen van het nieuwe vakgebied.



We moeten goed opletten bij het toepassen van rekenkundige operaties. Er is namelijk een verband tussen het type (gehele of gebroken) getallen dat als operand wordt gebruikt en het type *getalvorm* dat het resultaat zal krijgen. In onderstaande tabel is voor alle rekenkundige operaties dit verband gegeven, zoals dit in de meeste versies van BASIC optreedt.

Operatie	Operand 1	Operand 2	Resultaat
	geheel	geheel	gebroken
	geheel	gebroken	gebroken
↑	gebroken	geheel	gebroken
	gebroken	gebroken	gebroken
	geheel	geheel	geheel of gebroken (afhankelijk van de uitkomst)
	gebroken	geheel	gebroken
+, −, ∗, /	geheel	gebroken	gebroken
	gebroken	gebroken	gebroken

N.B. Vaak treden op dit punt verschillen op tussen de verschillende computers. Het is zinvol om hier de fabrieksdocumentatie op na te slaan.

Expressie
Een *expressie* (uitdrukking) in BASIC bestaat uit een groep symbolen, die samen een resultaat geven. Een expressie kunnen we eenvoudig vergelijken met het rechter deel van de formule, zoals bijvoorbeeld:

X = 4 * A + 6 * B − 7

De operanden, die in een expressie worden gebruikt, zijn *getallen, variabelen* en *functies.* Functies zullen in hoofdstuk 8 worden behandeld. Samen met operatoren kunnen de bovengenoemde operanden worden gecombineerd via een *rekenvoorschrift,* dat een of ander resultaat zal opleveren. De meest gebruikte soort expressie is de rekenkundige expressie.

Rekenkundige expressie
Een rekenkundige expressie stelt een *voorschrift* voor om een getal te berekenen, uitgaande van variabelen, constanten en rekenkundige operatoren.

Wiskundige notatie	BASIC notatie
x-y	X−Y
3b	3*B
z^2	Z↑2

Bij uitgebreide rekenkundige expressies is het van belang te weten hoe de expressie moet worden *uitgewerkt.* In BASIC zijn hiervoor een aantal regels vastgelegd die exact aangeven hoe de volgorde van handelen is.

Opmerking: Als in een BASIC programma de *waarde* van een expressie moet worden bepaald, houdt dit zonder meer in dat elke variabele een waarde moet hebben, anders kan de uitwerking niet worden uitgevoerd. Vaak wordt een variabele, als deze voor het eerst wordt gebruikt, gelijk gemaakt aan 0. Dit is natuurlijk niet van kracht als er een andere waarde aan wordt gegeven door het programma.

Prioriteiten
In de wiskunde hebben de operatoren een *prioriteit.* Dit wil zeggen, dat er een volgorde is volgens welke een expressie wordt uitgewerkt. De bekende regels in de wiskunde zijn: eerst machtsverheffen, dan vermenigvuldigen en delen, dan optellen en aftrekken. Deze prioriteitsregels gelden ook in de programmeertaal BASIC.

Prioriteit	Operatie	Notatie
Hoog ↑	Machtsverheffen	↑
	Vermenigvuldigen en delen	*,/
Laag	Optellen en aftrekken	+,−

Verdere regels die gelden bij het uitwerken bij expressies zijn:
regel 1
Operaties *binnen* haakjes worden uitgevoerd vóór operaties, die *niet* tussen haakjes staan.
regel 2
Operaties van *gelijke* prioriteit worden in volgorde van voorkomen van *links* naar *rechts* uitgevoerd.
regel 3
Operaties worden in *volgorde* van prioriteit uitgevoerd.

Voorbeeld: Werk de expressie A + 4 * B uit.

A + 4 * B 1 2	A en B zijn variabelen waarvan de waarde als eerste in de expressie moet worden ingevuld. Daarna komt het vermenigvuldigen van 4 en B omdat het vermenigvuldigen de hoogste prioriteit heeft. Het resultaat vervangt nu 4*B. Daarna wordt de optelling A + resultaat van de vorige bewerking uitgevoerd.
A + 4 * B	Stel: A = 3 en B = 6
3 + 4 * 6	Na invullen van de variabelen
3 + 24	Na vermenigvuldigen
27	Na optellen

Voorbeeld: Werk de expressie P * Q ↑ 3 + R uit.

P * Q ↑ 3 + R 1 2 3	P, Q en R zijn variabelen waarvoor als eerste de waarde in de expressie moet worden ingevuld. Nu moet van de expressie eerst Q tot de macht 3 worden verheven. Het resultaat moet dan met P worden vermenigvuldigd en de waarde van R moet er bij worden opgeteld.
P * Q ↑ 3 + R	Stel P = 2, Q = 3 en R = 5
2 * 3 ↑ 3 + 5	Na invullen van de variabelen
2 * 27 + 5	Na machtsverheffen
54 + 5	Na vermenigvuldigen
59	Na optellen.

Voorbeeld: Werk de expressie W * X − Y * Z uit.
Stel: W = 2, X = 3, Y = 1 en Z = 2

Expressie	
W * X − Y * Z	
2 * 3 − 1 * 2	Na in vullen van de variabelen
6 − 1 * 2	Na vermenigvuldigen, links beginnend
6 − 2	Na vermenigvuldigen
4	Na aftrekken

Probeer zelf eens een aantal getallen in te vullen in het onderstaande voorbeeld.

A * ((B + C) * D − E)

1

2

3

4

De getallen geven de prioriteiten

A * (B − C) − D / (E + F)

1

2

3

4

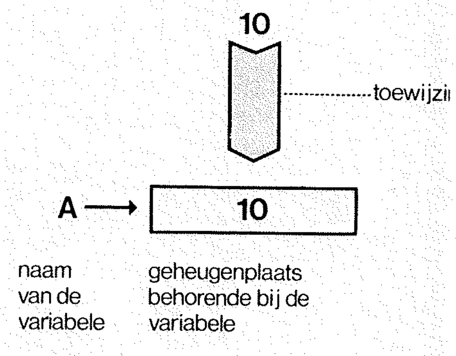
5

Toewijzingen

In de meeste programma's wordt de waarde, die beschikbaar is na het uitwerken van een expressie, toegewezen aan een variabele. Dit betekent dat de waarde van de expressie wordt *opgeslagen* in het geheugen, nl. op die plaats waar de betreffende variabele naar toe wijst. Het toewijzen van een waarde aan een variabele wordt als volgt genoteerd:

A = 10

Het = teken betekent hier de operatie toewijzen. Zie afbeelding 2.6.



Afb. 2.6. Toewijzing van een waarde aan een variabele.

We moeten de bovenstaande uitdrukking als volgt lezen:

A krijgt de waarde 10

of

A wordt gelijk gemaakt aan 10

(Verwar het = teken niet met het gelijkteken uit de wiskunde.)

We kunnen nu ook de volgende constructie krijgen:

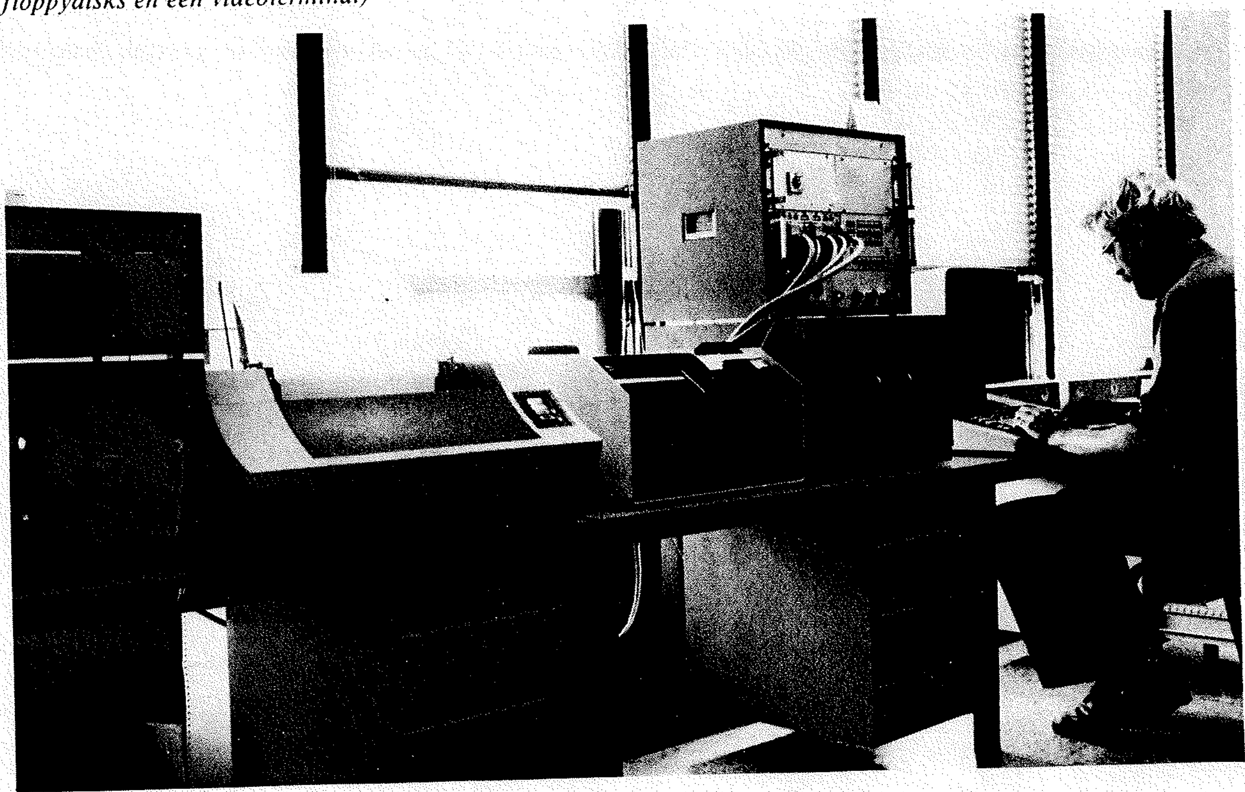
A = A + 7

Dit wil zeggen dat we eerst het *rechter lid* (A + 7) moeten uitwerken en daarna de waarde hiervan moeten toewijzen aan A. We moeten dus eerst de waarde van A uit het geheugen ophalen en daar 7 bij optellen. Daarna moeten we het *resultaat* opslaan op de geheugenplaats, die door A wordt aangewezen.

Samenvatting hoofdstuk 2

In dit hoofdstuk is behandeld hoe een programma wordt ingevoerd en gestart. De basisbegrippen zoals statement, variabelen, expressies, operatoren, prioriteiten en dergelijke die in hogere talen vaak voorkomen, zijn besproken. We hebben ons hier vooral gericht op eigenschappen zoals deze in BASIC voorkomen.

Een student op de TH Eindhoven tijdens de uitvoering van een opdracht aan een microcomputer. De foto toont dat de computer (kast waaruit de kabels komen) in het niet valt tegenover de randapparatuur (v.l.n.r. ponsbandponser met daarop een ponsbandlezer, printer, kaartlezer, twee floppydisks en een videoterminal)



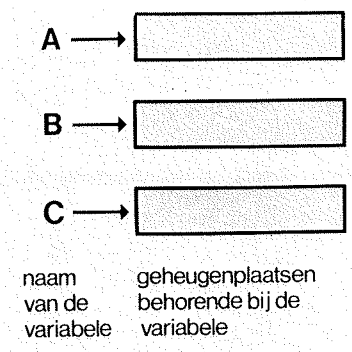
Hoofdstuk 3

Eenvoudig BASIC

In dit hoofdstuk worden enkele eenvoudige statements besproken. Dit zijn achtereenvolgens LET, PRINT, INPUT, GOTO en REM. Om met deze statements te kunnen werken, moeten we bovendien weten hoe we een programma kunnen starten, onderbreken, doorstarten en afdrukken.
Voor in het begin zal men vaak typefouten maken. Daarom wordt ook in dit hoofdstuk besproken hoe deze fouten kunnen worden gecorrigeerd.

3.1. Assignment statement: LET

Het LET statement wordt gebruikt om aan een variabele een numerieke of alfanumerieke *waarde* toe te kennen. De naam van het LET statement is dan ook assignment statement (assignment = toewijzing).
De toe te wijzen waarde mag zowel een *constante* als de waarde van een *expressie* zijn.
In het geheugen van de computer wordt voor elke variabele, die wordt gebruikt, een *ruimte* gereserveerd. Via de naam van de variabele wordt naar de geheugenplaats van de variabele gewezen.
Afbeelding 3.1 laat dit zien voor drie variabelen A, B en C. Via het toewijzing-statement (LET) wordt de waarde van de variabele in het geheugen ingevuld op de gereserveerde geheugenplaatsen.

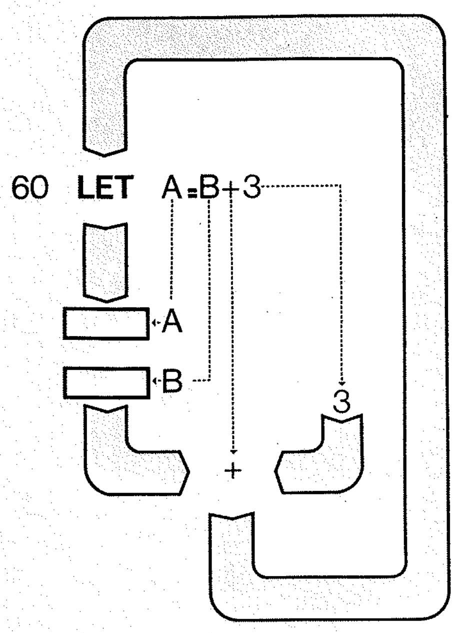


Afb 3.1. Variabelen zijn gekoppeld aan geheugenplaatsen.

Afbeelding 3.2 laat een programma zien dat bestaat uit een reeks LET statements.
Bij elke van de LET statements uit het programma krijgt de variabele, die links van het gelijkteken (=) staat, de waarde van het rechter

```
*LIST
110    LET X=10
120    LET X=X+1
130    LET Y=6
140    LET Z2=3*X+2*Y
150    LET W=3.1416*Z2↑2
160    Y=6
196    LET A$="TELEAC "
197    LET B$="CURSUS "
198    LET C$="MICRØPRØCESSØRS 2."
999    END
```

Afb. 3.2. Voorbeeldprogramma met LET statements.



Afb. 3.3. Werking van een LET statement.

lid. Zie ook de schematische voorstelling van afbeelding 3.3. Hierin wordt eerst de waarde van het rechter lid bepaald. Deze waarde wordt daarna aan de variabele toegewezen. In regel 110 krijgt de variabele X de waarde 10 (constante). In regel 120 krijgt X de waarde van het rechter lid, dat wil zeggen de waarde van X + 1, wat gelijk is aan 10 + 1 ofwel 11. Deze waarde wordt daarna aan het linker lid toegewezen. Dit wil dan weer zeggen dat op de geheugenplaats aangewezen door de naam X, het getal 11 wordt opgeslagen.
In regel 140 is het rechter lid:

```
3 * X  + 2 * Y
3 * 11 + 2 * 6
--------
  33   + 2 * 6
  33   +  12
------------
     45
```

De variabele Z2 wordt dus nu gelijk gemaakt aan 45.
In regel 150 wordt W gelijk gemaakt aan:

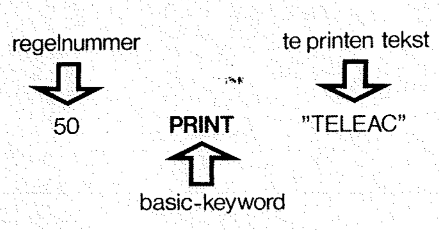
```
3.1416 * Z2↑2
3.1416 * 45↑2
3.1416 * 2025
-------------
 6361.73
```

Opmerking: ↑ betekent machtsverheffen.

In veel systemen is het niet nodig om het keyword LET te gebruiken bij het assignment statement. Het is namelijk al duidelijk dat er van een toewijzing sprake is doordat het gelijkteken in het statement voorkomt.
Zo heeft bijvoorbeeld regel 160 hetzelfde effect als regel 130.
Behalve de toewijzing van *numerieke* waarden (getallen) kunnen we ook *alfanumerieke* waarden in een assignment statement gebruiken. De regels 196, 197 en 198 geven hiervan een voorbeeld.
De alfanumerieke constanten moeten hierbij tussen *aanhalingstekens* (") staan en de variabele moet een alfanumerieke variabele zijn (dus de variabele naam moet als laatste letter het dollarteken ($) hebben).

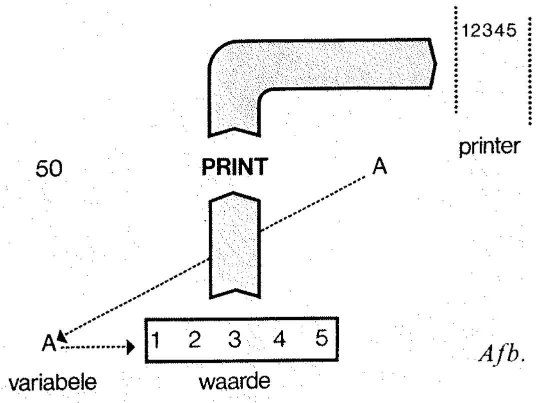
3.2. Output statement: PRINT

Met behulp van het PRINT statement wordt aan de gebruiker de mogelijkheid geboden om numerieke en alfanumerieke data vanuit de computer uit te voeren naar de *systeemterminal*. Als uitvoermedium wordt in BASIC meestal de *beeldbuis* van een videoterminal of *printer* van een teletype gebruikt. In afbeelding 3.4 wordt de opbouw van een PRINT statement geschetst. Afbeelding 3.5 toont de werking van het PRINT statement. Het PRINT statement haalt de te printen waarde uit het geheugen en verzorgt het afdrukken ervan. Na het keyword PRINT volgt een lijst van uit te voeren gegevens ofwel van verwijzingen naar de gegevens. De uit te voeren data kun-



Afb. 3.4. Opbouw van het PRINT statement.

nen getallen zijn, tekst, de waarde van expressies of de waarde van variabelen. De grootheden of de verwijzingen moeten worden gescheiden door komma's (,) of puntkomma's (;).



Afb. 3.5. Werking van het PRINT statement.

In afbeelding 3.6 worden de waarden, die in het programma van afbeelding 3.2 in de LET statements worden toegewezen, afgedrukt. Bovendien wordt in het voorbeeld nog een PRINT statement gegeven met daarbij rekenkundige bewerkingen in de lijst met af te drukken gegevens.
In regel 210 wordt de waarde van de constante (10) uitgevoerd. In regel 220 wordt achtereenvolgens de waarde van X, Y, Z2 en W uitgevoerd. Vaak zullen we zowel gegevens als teksten uitvoeren. Regels 230 en 240 geven hiervan een voorbeeld. Er wordt nu achtereenvolgens tekst, waarde van de variabele, tekst, enz. uitgevoerd. De uit te voeren tekst wordt tussen aanhalingstekens (") gezet om aan te geven dat het hier om een alfanumerieke constante gaat.
In regel 260 vinden we dat een alfanumerieke constante (NAAM:) en de waarde van de alfanumerieke variabele (A$) moeten worden uitgevoerd.
In de bovengenoemde voorbeelden is in de lijst van af te drukken variabelen (of grootheden) zowel de komma als de puntkomma als scheidingsteken (*terminator*) gebruikt. De functie van dit teken is globaal als volgt te verklaren. Als er een komma wordt toegepast, zal voor de uit te voeren waarde een *groot* veld worden gebruikt. Een *groot* veld bevat 14 posities. Bij gebruik van een puntkomma krijgen we voor de uit te voeren waarde een *klein* veld. Dit betekent dat er in het eerste geval meer *plaatsen* (posities op het papier of het scherm) zullen worden gebruikt voor de afzonderlijke gegevens dan in het tweede geval.
Bij het *kleine* veld worden de output-resultaten tegen elkaar gedrukt. Als we getallen uitvoeren die positief zijn, wordt het plusteken (+) niet afgedrukt maar er ontstaat een spatie. Omdat er nogal veel verschil is tussen verschillende soorten computers, zullen we hier niet al te diep ingaan op de *veldgrootte*. De fabrieksdocumentatie zal hier verder helpen. Het is zinvol om zelf enkele varianten uit te proberen.
Om de uitvoer van een programma beter leesbaar te maken worden soms *blanco regels* tussengevoegd. Dit wil zeggen dat we een regel overslaan. Door in het PRINT statement de lijst weg te laten, zal door de computer in de output een regel worden overgeslagen. Regel 280

van afbeelding 3.6 geeft hiervan een voorbeeld. Na elke regel waarin het PRINT keyword voorkomt zal automatisch worden overgegaan naar een nieuwe regel. Ook hier kunnen we onderuit komen, namelijk als we het PRINT statement afsluiten met een komma of puntkomma. Dan zal niet op de volgende regel worden verdergegaan, maar op dezelfde regel. Zie regels 290, 291 en 292 van het voorbeeld. We kunnen nog opmerken dat het *kleine* veld in geval van alfanumerieke output resulteert in het tegen elkaar aanschuiven van de *characters*.

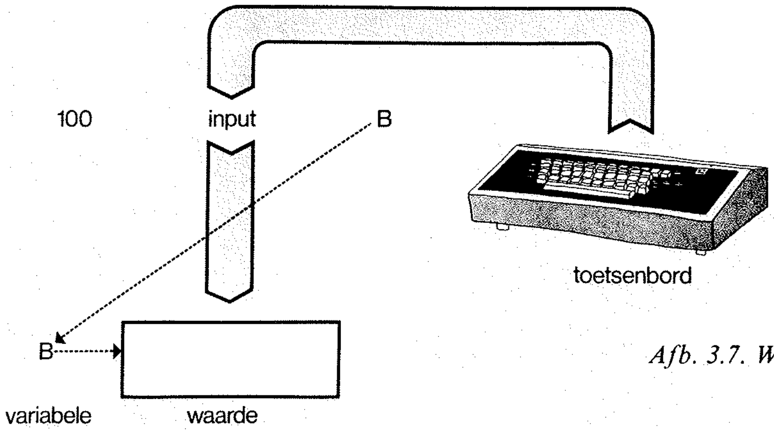
```
*LIST
110  LET X=10
120  LET X=X+1
130  LET Y=6
140  LET Z2=3*X+2*Y
150  LET W=3.1416*Z2↑2
160  Y=6
196  LET A$="TELEAC "
197  LET B$="CURSUS "
198  LET C$="MICRØPRØCESSØRS 2."
210  PRINT 10
220  PRINT X,Y,Z2,W
230  PRINT "X = ";X
240  PRINT "X = ";X,"Y = ";Y,"Z2 = ";Z2
250  PRINT A$
260  PRINT "NAAM : ";A$
270  PRINT 3*X,3*Y,3*Z2
280  PRINT
290  PRINT A$;
291  PRINT B$;
292  PRINT C$
999  END
*RUN
 10
 11          6           45              6361.74
X =  11
X =  11      Y =  6      Z2 =  45
TELEAC
NAAM : TELEAC
 33          18          135

TELEAC CURSUS MICRØPRØCESSØRS 2.
END PRØGRAM
```

Afb. 3.6. Voorbeeldprogramma met het PRINT statement.

3.3. Input statement: INPUT

Het INPUT statement wordt in BASIC gebruikt om tijdens de executie van een programma numerieke of alfanumerieke data vanaf de systeemterminal in te voeren. Afbeelding 3.7 laat de werking zien. Na het keyword INPUT zal een lijst volgen met namen van variabelen, waaraan door middel van invoerdata, die op het toetsenbord van de systeemterminal wordt ingetikt, een waarde worden toegewezen.



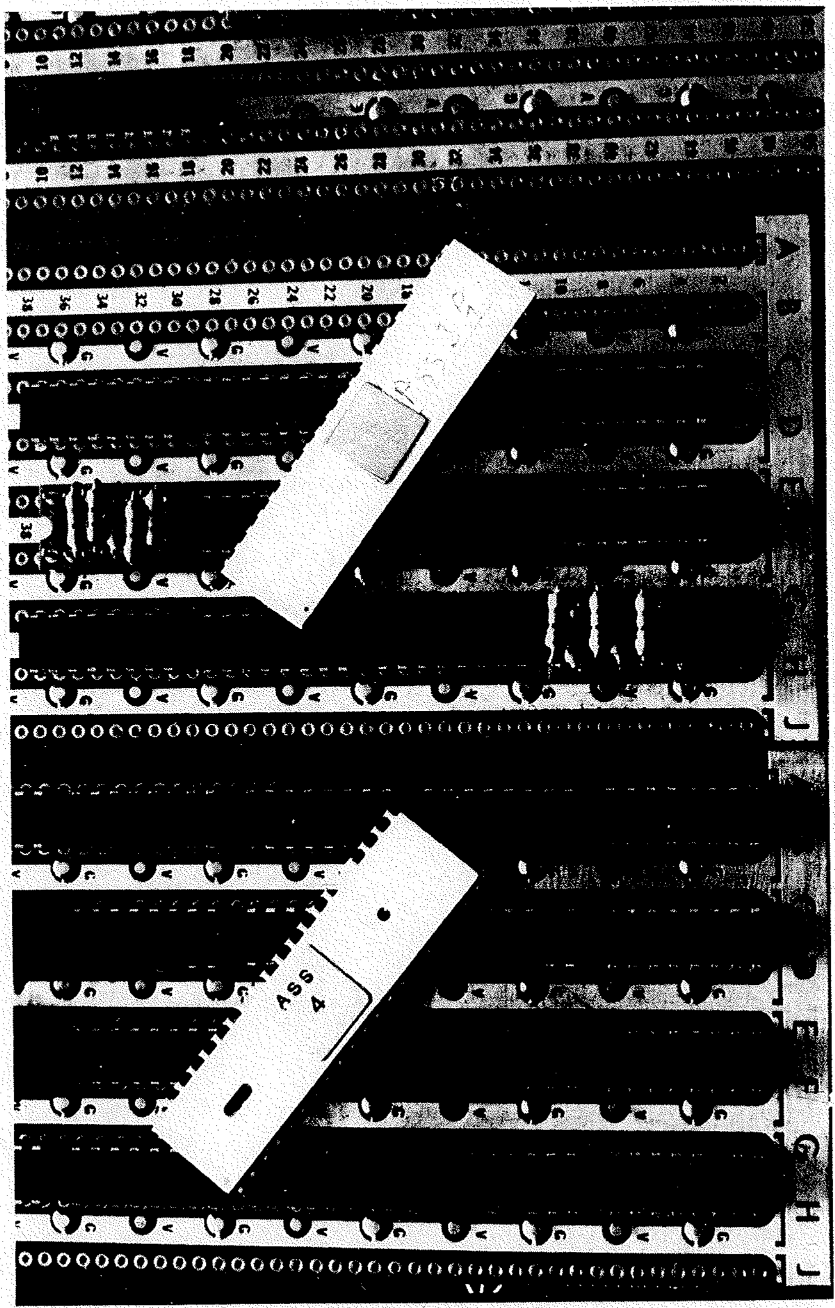
Afb. 3.7. Werking van het INPUT statement.

Tijdens de executie van een programma zal op de systeemterminal een *vraagteken* zichtbaar worden, zodra een INPUT statement wordt uitgevoerd. De beurt is nu aan de gebruiker. De computer wacht totdat de gebruiker een getal heeft ingetikt en zal de waarde toekennen aan de betreffende variabele.
Afbeelding 3.8 laat een programma zien met een aantal *varianten* van het INPUT statement.
In regel 300 moet de waarde, die de gebruiker intikt, worden toegewezen aan de variabele A. Bij regel 310 wordt om drie getallen gevraagd voor achtereenvolgens A, B en C. Bij het invoeren van de getallen mogen we de getallen op een regel intikken, er moet dan wel gebruik worden gemaakt van de komma als scheidingsteken tussen de drie getallen. Een tweede mogelijkheid bij dit statement is om na elk getal een RETURN te geven. De computer komt dan weer terug

```
*LIST
300   INPUT A :PRINT
305   PRINT "A = ",A
310   INPUT A,B,C:PRINT
315   PRINT "A = ";A,"B = ";B,"C = ";C
320   INPUT P,Q$,R$,Z:PRINT
325   PRINT "P = ",P
326   PRINT "STRING Q$ = ",Q$
327   PRINT "STRING R$ = ",R$
328   PRINT "Z = ",Z
999   END
*RUN
?123.456
A =           123.456
?12,34?567
A =  12       B =  34      C =  567
?567
?"TEKST","TWEEDE TEKST",98765
P =           567
STRING Q$ =               TEKST
STRING R$ =               TWEEDE TEKST
Z =           98765
END PRØGRAM
```

Afb. 3.8. Voorbeeldprogramma met het INPUT statement.

met een vraagteken voor de waarde van de tweede en derde variabele.
In regel 320 wordt als eerste om een numerieke waarde gevraagd, daarna om twee alfanumerieke waarde en daarna om een numerieke waarde. Wanneer we de alfanumerieke waarde invoeren, moeten we deze omsluiten door aanhalingstekens ("). Zie de output van het programma in afbeelding 3.8.
De Teleac/DAI-computer wijkt op dit punt af van de gebruikelijke definitie van INPUT. Gewoonlijk zal na het intypen van de gevraagde waarde de computer op de systeemterminal naar het begin



Een prototype van een associatief geheugen. (Vakgroep EB TH Eindhoven)

van de volgende regel gaan. Bij de Teleac/DAI-computer blijft de computer op dezelfde regel. Op deze wijze kunnen via INPUT meer vragen op dezelfde regel worden gesteld. Dit heeft enige voordelen in het gebruik van regels, maar heeft als groot nadeel het feit dat wordt afgewezen van de gebruikelijke BASIC versies. Wanneer nu in de Teleac/DAI-computer hetzelfde effect bereikt dient te worden als bijv. INPUT A op de meeste personal computers geeft moet nu gekozen worden INPUT A : PRINT als statement. Het statement: PRINT op dezelfde regel forceert de computer nu naar het begin van de volgende regel te gaan.

3.4. Voorbeelden met LET, INPUT en PRINT

Opdracht 1
Bepaal van een willekeurig in te voeren getal de 2de macht, de 3de macht, de 4de macht en de wortel. Druk dit in de tabel af. Afbeelding 3.9 laat een oplossing van de opdracht zien.

Opdracht 2
Bepaal de waarde van X tot de macht Y, waarbij X en Y willekeurig in te voeren getallen zijn.
Afbeelding 3.10 laat een oplossing van de opdracht zien.

Opdracht 3
Maak een programma dat uw naam vraagt en deze dan afdrukt.
Afbeelding 3.11 laat een oplossing van de opdracht zien.

```
*LIST
10    INPUT "VØER A IN";A:PRINT
20    PRINT
30    PRINT "A","A↑2","A↑3","A↑4","A↑.5"
40    PRINT
50    PRINT A,A↑2,A↑3,A↑4,A↑0.5
60    END
*RUN
VØER A IN
?3

A          A↑2         A↑3         A↑4         A↑.5

 3          9           27          81          1.73205
END PRØGRAM
*RUN
VØER A IN
?2

A          A↑2         A↑3         A↑4         A↑.5

 2          3.99995     7.9999      15.9998     1.41421
END PRØGRAM
```

Afb. 3.9. Programmavoorbeeld 1.

Opdracht 4
Maak een programma dat de inhoud van een vat berekent in liters. De diameter en hoogte van het vat worden ingevoerd in meters. Afbeelding 3.12 laat een oplossing van de opdracht zien.

Opmerking: In sommige van de programma's bevat een regel meerdere statements. De statements moeten hierbij van elkaar worden gescheiden door een dubbele punt (:).

```
*LIST
10    INPUT "VØER IN X EN Y";X,Y:PRINT
20    PRINT X;"↑";Y;" = ";X↑Y
30    END
*RUN
VØER IN X EN Y
?3
?3
 3↑ 3 =  27
END PRØGRAM
```

Afb. 3.10. Programmavoorbeeld 2.

```
*LIST
10    INPUT "HØE HEET U?";N$
20    PRINT "IK KEN U. U HEET: ";N$
30    END
*RUN
HØE HEET U?
?JAN
IK KEN U. U HEET: JAN
END PRØGRAM
```

Afb. 3.11. Programmavoorbeeld 3.

```
*LIST
10    PRINT "BEREKENING VAN DE INHØUD VAN EEN VAT.":PRINT
20    PRINT "VØER DE HØØGTE EN DE DIAMETER IN.":PRINT
30    INPUT "HØØGTE IN METERS =";H:PRINT
40    INPUT "DIAMETER IN METERS =";D:PRINT
50    LET I=1000*H*3.14159*(D/2)↑2:PRINT
60    PRINT "DE INHØUD IN LITERS IS : ";I;" LITER."
70    END
*RUN
BEREKENING VAN DE INHØUD VAN EEN VAT.

VØER DE HØØGTE EN DE DIAMETER IN.

HØØGTE IN METERS =
?0.6
DIAMETER IN METERS =
?0.4

DE INHØUD IN LITERS IS :  75.3982 LITER.
END PRØGRAM
*
```

Afb. 3.12. Programmavoorbeeld 4.

Statement	Effect
LET	Aan een variabele wordt een waarde toegekend.

Algemene vorm:
regelnumer [LET] v = x

waarbij geldt:
- x is een expressie of een alfanumerieke of numerieke variabele of een alfanumerieke of numerieke constante.
- v is een numerieke of alfanumerieke variabele.

NB. Het hakenpaar [ ] om het BASIC keyword LET betekent dat de tekst LET niet hoeft te worden gebruikt. Deze haken komen nooit voor in een BASIC statement zelf.

Effect:
Het LET statement kent de waarde van x toe aan de variabele v. Dit wil zeggen dat de variabele, die links van het gelijkteken staat, de waarde krijgt van het rechter lid.
Hierbij geldt echter wel dat de waarde van het rechter lid van dezelfde soort moet zijn als de variabele in het linker lid.

Statement	Effect
PRINT	De in het PRINT statement gedefinieerde tekst wordt op de systeemterminal zichtbaar gemaakt of afgedrukt.

Algemene vorm:
Regelnummer PRINT $f_1\ t_1\ f_2 \ldots t_2\ f_3\ t_3 \ldots f_n\ t_n$

waarbij geldt:
- f is een expressie, een numerieke of alfanumerieke variabele, een numerieke of alfanumerieke constante of leeg.
- t is een zogenaamd scheidingsteken. Bij de komma (,) als scheidingsteken wordt voor de uitvoer een groot veld genomen. Bij de puntkomma (;) een klein veld.

Effect:
Het PRINT statement zet de waarde van de uit te voeren grootheid om in leesbare output en plaatst deze, afhankelijk van de scheidingstekens, op bepaalde velden van de systeemterminal.
Als in het PRINT statement het laatste scheidingsteken ontbreekt, wordt na het printen automatisch overgegaan op een nieuwe regel.

N.B. Als f leeg is, wordt er voor dat bepaalde veld niets (spaties) afgedrukt.

Statement	Effect
INPUT	Door middel van het INPUT statement kan vanuit het keyboard van de systeemterminal data naar het programma worden doorgegeven.

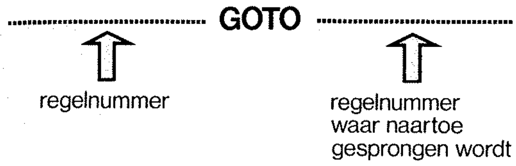
Algemene vorm:
Regelnummer INPUT c ; $v_1, v_2, \ldots, v_n$

waarbij geldt:
c is een alfanumerieke constante (tekst d.w.z. tussen aanhalingstekens (") geplaatst) of een alfanumerieke variabele.
$v_1 \ldots v_n$ zijn numerieke of alfanumerieke variabelen.

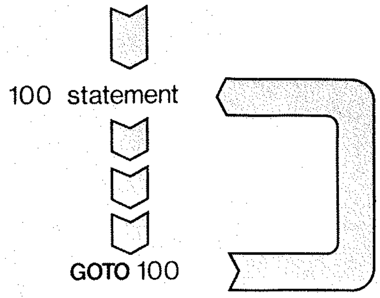
Effect:
Wanneer een INPUT statement in een programma voorkomt, zal bij de executie ervan door de computer een tekst c worden afgedrukt, die wordt gevolgd door een vraagteken (?).
De computer wacht daarna op akties van de gebruiker. Deze moet nu namelijk data invoeren. Bij het invoeren van data moet het type van de data overeenkomen met het type van de variabele waaraan de data wordt toegewezen. Indien er meer dan een waarde moet worden ingevoerd, mag de data op één regel staan, hierbij moeten komma's als scheidingsteken worden gebruikt. De data mag ook op verschillende regels worden ingevoerd. De computer zal door middel van vraagtekens steeds om data blijven vragen, totdat voor elke variabele een toewijzing heeft plaats gehad.

3.5. Niet voorwaardelijke sprong: GOTO

Het GOTO statement biedt ons de mogelijkheid om de normale *volgorde* van executie van statements te doorbreken. Bij een BASIC programma zal de executie van het statement plaatsvinden in volgorde van regelnummers. Wanneer we een programma of deel van een programma een aantal keren achter elkaar zouden willen uitvoeren, zou dit betekenen dat we telkens het RUN commando zouden moeten geven. Een andere oplossing zou kunnen zijn, de reeks statements net zo vaak in het programma op te nemen als we het programma willen uitvoeren. Beide oplossingen zijn overbodig als we het GOTO statement toepassen. Het GOTO statement laat namelijk de computer een niet-conditionele (onvoorwaardelijke) sprong uitvoeren naar een opgegeven regelnummer. Als we hiervoor het begin van het programma kiezen, houdt dit in dat we het programma oneindig vaak kunnen uitvoeren zonder het steeds op te starten of het onnodig lang te maken.
Afbeelding 3.13 toont de opbouw van het GOTO statement en afbeelding 3.14 laat de werking zien.
Afbeelding 3.15 laat een aangepaste vorm zien van voorbeeld 3.



Afb. 3.13. Opbouw van het GOTO statement.



Afb. 3.14. Werking van het GOTO statement.

> *Opmerking*
> Op uw terminal (scherm) verschijnt het voorbeeldprogramma uiteraard in één kolom, om ruimte te besparen zijn verschillende voorbeeldprogramma's over twee kolommen geplaatst.
> De kolommen zijn gescheiden door een verticale lijn.

```
*LIST
10     PRINT "HØE HEET U?"
20     INPUT N$
30     PRINT "IK KEN U, U HEET : ";N$
40     GØTØ 10
50     END
*RUN
HØE HEET U?
?JAN
IK KEN U, U HEET : JAN
HØE HEET U?
?PIET
IK KEN U, U HEET : PIET
HØE HEET U?
?KLAAS
IK KEN U, U HEET : KLAAS
HØE HEET U?
?! <—— BREAK
```

(vervolg programma)

```
BREAK IN LINE 20
*CØNTINUE
?MARIE
IK KEN U, U HEET : MARIE
HØE HEET U?
?! <—— BREAK

BREAK IN LINE 20
*
```

Afb. 3.15. Voorbeeldprogramma met het GOTO statement.

PAS OP!
Deze manier van programmeren betekent wel dat we de computer in een oneindige lus hebben gebracht. Hij zal namelijk uit zichzelf nooit meer dit programma kunnen verlaten. Dit is vaak het resultaat van een *programmafout*. De gebruiker kan echter wel het programma afbreken door de BREAK toets in te drukken. Hierdoor breekt de computer de executie van het programma af en komt vrij voor nieuwe commando's.
Bij het afbreken van het programma zal de computer de volgende boodschap geven:

BREAK IN LINE...

Door het commando CONT (continue) in te tikken, kan het programma weer worden doorgestart op het punt waar het was onderbroken.
In afbeelding 3.15 zien we dat het programma wordt onderbroken en daarna weer wordt doorgestart.
De BREAK toets kan altijd worden gebruikt om een programma te onderbreken. Zie hiervoor de betreffende fabriekshandleidingen. Soms komen ook andere letters voor.

Statement	Effect
GOTO	Spring naar een gedefinieerd regelnummer

Algemene vorm:
regelnummer GOTO n

waarbij geldt:
n is een regelnummer van het programma.

Effect:
Het GOTO statement is een *niet-conditionele* sprongopdracht, waarbij de voortgang in het programma zal plaatsvinden bij de regel met het regelnummer dat volgt op het keywoord GOTO. De normale sequentie van afhandeling volgens oplopende regelnummers van statements wordt hierbij doorbroken.

3.6. Correcties van typefouten

Meestal zal een gebruiker van een (personal) computer niet foutloos de BASIC statements kunnen intikken. Zelfs de beste typiste maakt wel eens een foutje. Bovendien zal het nogal eens voorkomen, dat men *denkfouten* maakt bij het opzetten van een programma. Het resultaat in beide gevallen is dat we de programmatekst moeten verbeteren. Voor deze correcties bezit BASIC enkele standaardtechnieken.

Als we bezig zijn met het intikken van een BASIC regel, is het mogelijk door middel van het aanslaan van de CHAR DEL toets (character delete = karakter weglaten) de *laatste* letter, die we op een regel hebben ingetikt, te verwijderen.

Dit geeft ons de mogelijkheid om van rechts naar links letter voor letter weg te laten. Als we op het punt van de tikfout zijn aangekomen, kunnen we de juiste letters intikken en de fout is dan verbeterd. Het betekent dus dat we de regel tot en met de fout 'schoonmaken' en daarna op het lege stuk de rest invullen. Afbeelding 3.16 laat zien wat er telkens gebeurt. (Alle regels die zijn afgedrukt liggen op dezelfde plaats.)

```
*NEW
*10 PIRNT ← CHAR DEL
*10 PIRN  ← CHAR DEL
*10 PIR   ← CHAR DEL
*10 PI    ← CHAR DEL
*10 PRINT
```

Afb. 3.16. Correcties van tikfouten door middel van CHAR DEL.

De gegeven methode is goed bruikbaar voor *kleine* correcties tijdens de invoerfase. Als we echter te veel fouten hebben gemaakt, is het soms beter de regel helemaal weg te laten en opnieuw beginnen. Het weglaten van de regel, tijdens de invoerfase, kan geschieden door de BREAK toets in te drukken.

Wanneer we de tikfouten pas zien nadat we de regels al in het geheugen van de computer hebben opgeslagen, kunnen we de bovenstaande technieken niet meer toepassen. We kunnen nu de foutieve regels verbeteren door die regels voorafgegaan door het regelnummer opnieuw in te tikken. Als we overbodige regels willen verwijderen, kunnen we volstaan met het regelnummer, gevolgd door RETURN. De regel is daarna uit het geheugen verdwenen.

Soms zal het ook noodzakelijk zijn om regels tussen te voegen. De

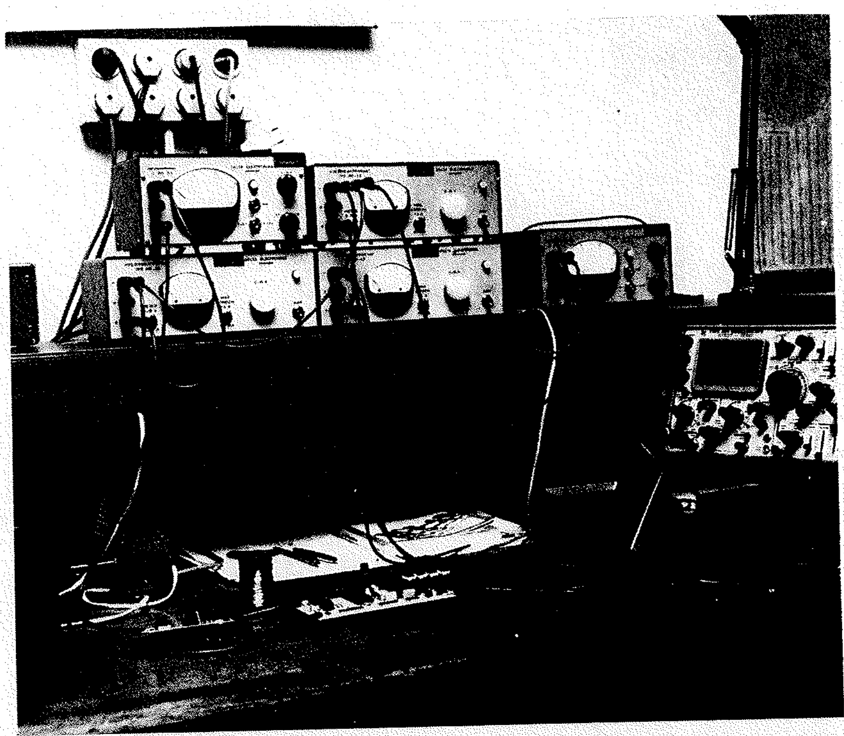
techniek hiervoor is om bij het opzetten van het programma de regels te nummeren in veelvouden van bijvoorbeeld 10 of 100. Het tussenvoegen is nu niets anders dan de nieuwe regels een regelnummer te geven, dat tussen de regels, waar tussen we de nieuwe regel willen plaatsen, in ligt.
Afbeelding 3.17 laat enkele van deze correcties zien. Regel 10 wordt met een tikfout ingevoerd. Er staat namelijk PRINTT i.p.v. PRINT.

```
*NEW
*10 PRINTT "HØE HEET U?"
10    *** PRINTT ?"HØE HEET U?"
SYNTAX ERRØR
*10 PRINT "HØE HEET U?"
*30 PRINT "IK KEN U NU, U HEET : ;N$
30    *** PRINT "IK KEN U NU, U HEET : ;N$?
SYNTAX ERRØR
*30 PRINT "IK KEN U NU, U HEET :";N$
*40 END
*LIST
10    PRINT "HØE HEET U?"
30    PRINT "IK KEN U NU, U HEET :";N$
40    END
*20 INPUT N$:PRINT
*LIST
10    PRINT "HØE HEET U?"
20    INPUT N$:PRINT
30    PRINT "IK KEN U NU, U HEET :";N$
40    END
*RUN
HØE HEET U??JAN
IK KEN U NU, U HEET :JAN
END PRØGRAM
```

Afb. 3.17. Correcties van een programma.

De computer herkent dit niet en geeft een *foutmelding*. Hierna voeren we regel 10 opnieuw in. De oude regel 10 verdwijnt nu. De invoer van regel 20 wordt vergeten. Na het LIST commando merken we dit. We tikken alsnog regel 20 in en geven weer het commando LIST. Het programma is nu goed.
De Teleac/DAI personal computer bevat bovendien nog de zogenaamde EDIT werkmode. In deze speciale mode kunnen we correcties in programma's aanbrengen. We tikken hiervoor het commando EDIT in. Na enige tijd verschijnt de programmatekst op het scherm. We kunnen nu door middel van de cursor (knipperend streepje) control toetsen (grijze toetsen met pijlen) naar de plaats in de programmatekst gaan waar we correcties willen aanbrengen. Nieuwe letters die we intikken worden tussengevoegd in de bestaande tekst. Ook is het mogelijk om letters door middel van de CHAR DEL toets weg te halen.
Vaak is het nuttig om samen met de cursor control toetsen de REPT (repeat) toets in te drukken. We gaan dan snel naar de plaats waar we willen corrigeren.
Door achtereenvolgens de toets BREAK en de spatiebalk in te druk-

ken worden de correcties geëffectueerd. Willen we de correcties echter ongedaan maken dan moeten we twee keer BREAK geven.
We verlaten in beide gevallen de EDIT mode. In de EDIT mode is het soms nuttig om in de plaats van cursor, de tekst te verplaatsen op het scherm. Dit is mogelijk door tegelijk met de cursor control toetsen de SHIFT toets in te drukken.



Een in ontwikkeling zijnd microcomputersysteem in een meetopstelling.

```
*LIST
10    PRINT "REGEL 1"
20    PRINT "REGEL 2"
30    PRINT "REGEL 3"
40    PRINT "REGEL 4"
50    PRINT "REGEL 5"
999   END
*LIST 20
20    PRINT "REGEL 2"
*LIST 30-
30    PRINT "REGEL 3"
40    PRINT "REGEL 4"
50    PRINT "REGEL 5"
999   END
*LIST -30
10    PRINT "REGEL 1"
20    PRINT "REGEL 2"
30    PRINT "REGEL 3"
*LIST 20-40
20    PRINT "REGEL 2"
30    PRINT "REGEL 3"
40    PRINT "REGEL 4"
*
```

Afb. 3.18. Voorbeelden van LIST commando's.

3.7. Commando: LIST

Door middel van het LIST commando is het mogelijk om de programmatekst, die in het geheugen staat, zichtbaar te maken. Er ontstaan echter problemen wanneer we zeer grote programma's hebben. Het 'listen' van het programma zou dan zeer lang duren. Bij gebruik van een *video-display* als outputterminal zal de tekst, die op het scherm zichtbaar kan worden gemaakt, twintig regels zijn. Hebben we meer regels in het programma, dan zal de tekst van onderen naar boven over het scherm schuiven. Het resultaat is dat we nooit de kans zullen krijgen om de programmatekst op ons gemak te bestuderen.
In veel gevallen zijn we slechts in een klein deel van het programma geïnteresseerd. Voor dit soort gevallen zijn de volgende *varianten* van het LIST commando beschikbaar.
Als we het 'listen' willen onderbreken (om zo tijd te sparen) kunnen we de BREAK toets indrukken. Het listen kan worden gestopt door de spatiebalk in te drukken. Het opnieuw indrukken start het listen weer, in geval van de Teleac/DAI personal computer.
Zie afbeelding 3.18. Deze spreekt verder voor zichzelf.

Commando	Effect
LIST	maakt het hele programma zichtbaar
LIST n	maakt regel n zichtbaar
LIST n-	maakt het programma vanaf regel n zichtbaar
LIST -m	maakt het programma tot en met regel m zichtbaar
LIST n-m	maakt het programma van regel n tot en met regel m zichtbaar (Hierbij zijn n en m regelnummers)

3.8. Commando: RUN

Door middel van het RUN commando starten we de *executie* van het programma dat in het geheugen staat. Van dit commando zijn twee varianten beschikbaar namelijk:

Commando	Effect
RUN	start de programma-executie op de eerste regel
RUN n	start de programma-executie bij regel n van het programma (hierbij is n een regelnummer)

Het tweede soort is erg handig als we niet het hele programma, maar een deel van het programma willen uitproberen. Zie ook afbeelding 3.19.

```
*LIST
10    PRINT "UITVØERING VAN REGEL 10"
20    PRINT "UITVØERING VAN REGEL 20"
30    PRINT "UITVØERING VAN REGEL 30"
40    END

*RUN
UITVØERING VAN REGEL 10
UITVØERING VAN REGEL 20
UITVØERING VAN REGEL 30
END PRØGRAM
*RUN 30
UITVØERING VAN REGEL 30
END PRØGRAM
*RUN 20
UITVØERING VAN REGEL 20
UITVØERING VAN REGEL 30
END PRØGRAM
```

(vervolg programma)

```
*RUN 10
UITVØERING VAN REGEL 10
UITVØERING VAN REGEL 20
UITVØERING VAN REGEL 30
END PRØGRAM
*
```

Afb. 3.19. Starten van programma's door middel van RUN commando's.

3.9. Commentaar statement: REM

In veel gevallen zal een programma bestaan uit een reeks *formules,* (verwoord via statements) die over het algemeen niet eenvoudig te begrijpen zijn. De variabelen, die hiervan deel uitmaken, zullen een betekenis hebben, die niet direct uit de naam van de variabele zal volgen.
Vaak zijn de programma's onderverdeeld in *deeltaken,* die vrijwel los van elkaar staan. Samenvattend mogen we concluderen, dat het erg nuttig zou zijn om tussen de verschillende statements van een programma een of meer regels commentaar op te nemen. Dit om de programmatekst nader toe te lichten.
Het is bijna van levensbelang om goed commentaar bij een programma te zetten om later nog te kunnen begrijpen wat het programma doet.
In BASIC is voor dit doel het REM statement opgenomen. REM is een afkorting van remark (opmerking). Het keyword REM mag worden gevolgd door willekeurige tekst. Het REM statement heeft geen enkele invloed op de afhandeling van het programma en is alleen maar toegevoegd om een en ander beter leesbaar te maken.
Het programma van afbeelding 3.20 wordt door toepassing van het commentaarstatement veel duidelijker.

```
*LIST
10    REM DIT PRØGRAMMA BEPAALT ENKELE MACHTEN VAN EEN
20    REM IN TE VØEREN GETAL
30    REM
40    REM ------------------------------------------
50    REM INTERN WØRDT MET HET GETAL A GEWERKT
60    REM ------------------------------------------
70    REM INVØERFASE
80    REM
90    INPUT A:PRINT
100   REM ------------------------------------------
110   REM UITVØERFASE
120   REM KØPREGEL
130   PRINT "A","A↑2","A↑3","A↑4"
140   PRINT
150   REM DATAREGEL
160   PRINT A,A↑2,A↑3,A↑4
170   REM
180   REM ------------------------------------------
190   REM EINDE VAN HET PRØGRAMMA
200   END
*RUN
?1.1111
A             A↑2           A↑3           A↑4

 1.1111        1.23454       1.3717        1.5241
END PRØGRAM
```

Afb. 3.20. Toevoegen van commentaar in een programma.

Samenvatting hoofdstuk 3

In dit hoofdstuk zijn de LET, PRINT, INPUT, REM en GOTO statements besproken. Aan de hand van een aantal voorbeelden is gedemonstreerd hoe met deze statements eenvoudige programma's kunnen worden gemaakt.
Om met deze statements te kunnen werken zijn enkele noodzakelijke commando's RUN, LIST en BREAK voor het bedienen van een personal computer besproken. Samen met de behandelde statements moet dit een voldoende basis zijn om aan de slag te kunnen.



Via een tweetal microcomputers worden op de TH te Eindhoven examens afgenomen. Omdat de computers niet op 'afkijken' kunnen testen is een aantal camera's met monitor aanwezig voor het voorkomen van fraude.

Multiple choice-vragen bij hoofdstuk 3

A. Welke zijn correcte BASIC constanten?

a 2,743
b 1/2
c 6E0,5
d 0,123 – E11

1 a en b
2 b
3 geen van alle
4 a, c en d

B. Welke zijn correcte identifiers?

a C2
b C"2"
c C1.2
d C'15'

1 a
2 b en d
3 a, b en c
4 a, c en d

C. **a** In BASIC mag een numerieke constante een komma bevatten.
b Een string mag geen cijfers bevatten.

1 uitspraak a is goed en uitspraak b is goed
2 uitspraak a is fout en uitspraak b is goed
3 uitspraak a is goed en uitspraak b is fout
4 uitspraak a is fout en uitspraak b is fout

D. Welke van de BASIC statements komen overeen met de ervoor staande formules?

a $s = \frac{1}{2} at^2$; 10 S = 0.5 * A * T * T
b $c = \sqrt{a^2 + b^2}$; 10 C = (A ↑ 2 + B ↑ 2) ↑ 2
c $r = (p - q)^{S-3}$; 10 R = (P − Q) ↑ S − 3
d $y = 7\,x^{1/6}$; 10 Y = (7.0 * X) ↑ (1.0/6.0)

(Opmerking: het getal 10 stelt het regelnummer in een programma voor.)

1 a, b en c
2 a, b en d
3 a
4 b

E. Bepaal de waarde van T1 door middel van het uitwerken van de volgende expressie:

T1 = A * B/2/C * D

met A = 2,0; B = 3.0; C = 4.0 en D = 5.0

De juiste waarde is:

1 0.15
2 3.75
3 60
4 de expressie is niet uit te werken

F. In de wiskunde kan een wortel x_1 van onderstaande kwadratische vergelijking:

$$ax^2 + bx + c = 0$$

worden bepaald volgens de formule:

$$x_1 = \frac{-b + \sqrt{b^2 - 4\,ac}}{2a}$$

We kiezen voor a, b, c en x_1 respectievelijk de BASIC variabelen A, B, C en X1.

Welke van de onderstaande statements bepaalt de waarde van x_1 op de juiste wijze?

```
1 10 LET X1 = (−B+(B ↑ 2−4 AC) ↑ .5) / (2 A)
2 10 LET X1 = −B+(B ↑ 2−4 * A * C) ↑ .5/2 * A
3 10 LET X1 = (−B+(B ↑ 2−4 * A * C) ↑ .5)/2 * A
4 10 LET X1 = (−B+(B ↑ 2−4 * A * C) ↑ .5)/(2 * A)
```

G. We willen in de formule:

$$y = 100\,(1 + x + x^2 + x^3)$$

de waarde van y bepalen. We kiezen voor x en y. respectievelijk de variabelen X en Y. Welk van de onderstaande oplossingen is fout?

```
1 10 LET   Y = 100 (1 + X + X ↑ 2 + X ↑ 3)
  20 END
2 10 LET   B = 100
  20 LET   Y = B * (1 + X + X ↑ 2 + X ↑ 3)
  30 END
3 10 LET   Y = 100 * (1 + X * (1 + X * (1 + X)))
  20 END
4 10 LET   Y = X
  20 LET   Y = Y + 1
  30 LET   Y = Y * X
  40 LET   Y = Y + 1
  50 LET   Y = Y * X
  60 LET   Y = Y + 1
  70 LET   Y = Y * 100
  80 END
```

H. a Door het statement

10 LET I = I + J

wordt de waarde van de variabele I verhoogd met de waarde van de variabele J als de waarde van de variabele J positief is.

b Door het statement

10 LET S$ = "9 AUG 1944".

wordt de tekst tussen aanhalingstekens in het geheugen gezet op de plaats, die wordt aangewezen door de variabele S$.

1 uitspraak a is goed en uitspraak b is goed
2 uitspraak a is fout en uitspraak b is goed
3 uitspraak a is goed en uitspraak b is fout
4 uitspraak a is fout en uitspraak b is fout

I. Om de waarde van drie variabelen (A, B en C) in een programma af te drukken, zijn vier deelprogramma's geschreven, namelijk:

a
```
110 PRINT A, B, C
120 PRINT
```
b
```
110 PRINT A,
120 PRINT B,
130 PRINT C,
140 PRINT
```
c
```
110 PRINT A,
120 PRINT B,
130 PRINT C,
140 PRINT
```
d
```
110 PRINT A, B,
120 PRINT C,
130 PRINT
```

Drie van de vier programma's hebben hetzelfde effect. Dit zijn:

1 a, b en c
2 a, b en d
3 a, c en d
4 b, c en d

J. Programma:

```
10 LET I = 0
20 LET I = I + 1
30 GOTO 10
40 LET I = 1
50 END
```

De variabele I:

1 wordt 0 en blijft 0
2 wordt achtereenvolgens: 0, 1, 0, 1, 0, 1 . . .
3 wordt achtereenvolgens: 0, 1, 1, 1, 1, . . .
4 wordt achtereenvolgens: 0, 1, 2, 2, 2, 2, . . .

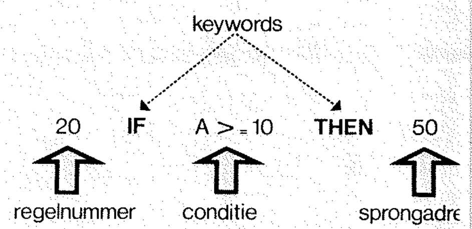
Hoofdstuk 4

Beslissingen in BASIC

In dit hoofdstuk wordt besproken hoe in een programma een vertakking kan worden gerealiseerd met behulp van een zogenaamde conditionele sprongopdracht. Bij de conditionele sprongopdracht worden condities getest. Bij deze tests worden relaties tussen grootheden bekeken. Afhankelijk van het resultaat zal de vervolgrichting in het programma worden bepaald.
De meeste beginnende programmeurs zullen in hun programma's fouten maken. Om deze fouten op te kunnen sporen zullen we enkele technieken aandragen.

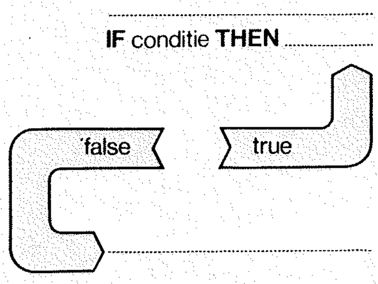
4.1. Conditionele sprongopdrachten

In de voorgaande hoofdstukken hebben we besproken dat de statements van een BASIC programma in het algemeen in volgorde van *regelnummer* zullen worden uitgevoerd. We moeten ons wel realiseren dat een computer pas echt van nut wordt wanneer we hem uitrusten met de mogelijkheid om voor ons *beslissingen* te nemen. Dit betekent niet dat de computer op eigen houtje zal kunnen bepalen wat er moet gebeuren. We moeten de computer uitrusten met de mogelijkheid om voor ons en op ons bevel een test uit te voeren. Afhankelijk van het resultaat van de test zal de computer dan in één van de door ons opgegeven mogelijke richtingen verder gaan met het programma. Deze mogelijkheid noemen we in de praktijk het uitvoeren van een *logische beslissing*. Deze naam is afkomstig van het oplossen van een logische expressie, wat hierbij zal moeten gebeuren. We zullen namelijk een of andere uitspraak doen. Afhankelijk van het 'true' of 'false' zijn van deze uitspraak zullen we in het programma al dan niet een *sprong* maken.

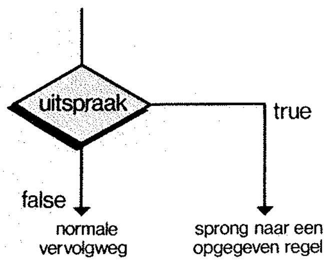


Afb. 4.1. Opbouw van het IF. THEN statement.

In het voorafgaande is de niet-conditionele sprong (GOTO) reeds aan de orde geweest. Hierbij werd altijd een sprong naar een opgegeven regelnummer uitgevoerd. Als we nu het uitvoeren van de sprong afhankelijk maken van de waarde van een in het statement opgenomen expressie, hebben we de gewenste voorwaardelijke functie mogelijk gemaakt.
De conditionele sprongopdracht wordt in BASIC gerealiseerd door middel van het IF.THEN statement (zie afbeelding 4.1). Na het keyword IF zal een *uitspraak* moeten staan. Afhankelijk van de waarde van de expressie wordt bepaald of we een sprong zullen uitvoeren naar het regelnummer dat volgt op het keyword THEN of dat we doorgaan op de volgende regel (zie ook afbeelding 4.2).



Afb. 4.2. Werking van het IF. THEN statement



Afb. 4.3. Symbool voor het IF. THEN statement

In afbeelding 4.3 zien we in de vorm van een *flow-diagram* wat de inhoud van een conditionele sprongopdracht is.
Als de uitspraak waar is (de logische waarde 'true' heeft), zal de sprong worden uitgevoerd. Als de uitspraak niet waar is ('false'), wordt normaal doorgegaan met de volgende regel van het programma.
In het programma van afbeelding 4.4 wordt een en ander duidelijk gemaakt. De conditionele sprongopdracht van regel 50 bevat de uitspraak A = B. Indien (IF) dit waar is, dan (THEN) wordt gesprongen naar regel 100. Als het niet waar is, gaan we normaal verder (hier op regel 60).

Afb. 4.4. Programmavoorbeeld met het IF. THEN statement

```
10 ...
20 ...
30 ...
40 ...
50 IF A=B THEN 100
60 REM HIER ALS A ØNGELIJK AAN B
70 ...
80 ...
90 ...
100 REM HIER ALS A GELIJK AAN B
```

In het IF. THEN statement kunnen we veel soorten uitspraken opnemen. Het resultaat van deze uitspraken zal altijd een *logische waarde* ('true' of 'false') moeten zijn. De uitspraken zeggen iets over de relatie tussen variabelen, die zowel van rekenkundige als van logische aard kan zijn. Een *rekenkundige relatie* zegt bijvoorbeeld dat de ene variabele groter is dan de andere of dat ze bijvoorbeeld gelijk zijn. *Logische relaties* zeggen dat zowel de ene als de andere uitspraak waar is. Voordat we verder kunnen gaan met BASIC moeten we eerst dieper op deze materie ingaan.

4.2. Relationele operatoren

De relationele operatoren worden gebruikt om een uitspraak te doen over de relatie tussen twee grootheden. Zo kan worden onderzocht of het ene getal groter is dan het andere. De relationele operatoren werken op *numerieke* operanden (dus. getallen). Ze geven, afhankelijk van de relatietest, als resultaat de logische waarde 'true' of 'false'. In de relatietest wordt een uitspraak gedaan. Deze uitspraak zal waar of niet waar zijn. De relationele operatoren, die voorkomen in BASIC, zijn:

Operator	Operatie	Voorbeeld	
=	is gelijk aan	A = B	True als de waarde van A gelijk is aan de waarde van B. Anders false.
<	is kleiner dan	P < Q	True als de waarde van P kleiner is dan de waarde van Q. Anders false.

Operator	Operatie	Voorbeeld	
>	is groter dan	Z>A	True als de waarde van Z groter is dan de waarde van A. Anders false.
< =	is kleiner of gelijk aan	Z1 < = P	True als de waarde van Z1 kleiner is dan P of gelijk is aan P. Anders false.
> =	is groter of gelijk aan	P2 > = Q	True als de waarde van P2 groter of gelijk aan Q is. Anders false.
< >	is ongelijk aan	B < > P	True als de waarde van B niet gelijk is aan de waarde van P. Anders false.

4.3. Relationele expressies

Naast de reeds besproken expressies, die met behulp van rekenkundige operatoren werken, zijn er ook *expressies* die relationale operatoren bevatten. Na uitwerking van dit soort expressies is een logische waarde het resultaat. De twee mogelijke logische waarden zijn true en false. In de relationele expressies wordt een uitspraak gedaan over de relaties van de operanden. Zo'n uitspraak kan waar of niet waar zijn.

In plaats van de naam logische waarde gebruikt men ook vaak de naam *booleaanse* waarde.

Relationele expressie	Betekenis
A > B	De uitspraak is dat A groter is dan B.
P < > Q	De uitspraak is dat P ongelijk is aan Q.
C = D	De uitspraak is dat C gelijk is aan D.

Afhankelijk van de waarde van de variabelen zal zo'n uitspraak al dan niet waar zijn.

Voorbeeld
Stel: A = 3, B = 7, P = 6, Q = 4, C = 10 en D = 10

Relationele expressie	Waarde van het resultaat
A > B	3 < 7 : false
P < > Q	6 < > 4: true
C = D	10 = 10 : true

De relationele expressies zullen in programma's veel worden gebruikt om te testen of bepaalde beweringen waar zijn. Daarna kunnen we dan afhankelijk van het testresultaat in de een of andere richting verder gaan (zie het programma van afbeelding 4.4).

Opmerking 1
Het is niet gebruikelijk om een relationele expressie meer dan twee tests achter elkaar te laten bevatten. Vaak is dan namelijk het resultaat niet gedefinieerd. De waarde van de expressie: A > B > C is niet gedefinieerd.

Opmerking 2
Bij het toepassen van relationele operatoren moeten we goed oppassen. De computer zal namelijk intern werken met benaderingen van getallen. Er wordt in de machine een *binaire code* voor de getallen gebruikt. Ook zal een integer en een floating point representatie van een zelfde getal intern niet dezelfde code bezitten. Dit betekent dat het testen op gelijkheid soms *fouten* zal introduceren. Daarom kunnen we beter testen op bij voorbeeld het groter of gelijk zijn dan op gelijkheid alleen. Het testen op nul is om dezelfde reden gevaarlijk. De nauwkeurigheid van de interne rekenprocedures bepaalt hoe men moet werken.
Het is noodzakelijk om hierbij de *fabrieksdocumentatie* goed te bestuderen.

4.4. Logische operatoren en logische expressies

Op de waarde van de relationele expressie (die gelijk aan true of false is) kunnen we een aantal logische operaties uitvoeren. Deze logische operaties leveren als resultaat weer een logische waarde op, dus *true* of *false*.
De logische operatoren, die in de meeste versies van BASIC voorkomen, zijn:

- AND
- OR
- NOT

Met behulp van combinaties van de bovengenoemde logische operatoren kunnen we bijvoorbeeld testen of de bewering waar is dat A groter is dan B en A kleiner dan C(A > B AND A < C). Het resultaat zal de waarde true hebben als zowel de eerste als de tweede uitspraak waar is. Een andere bewering zou kunnen zijn: A is kleiner dan C of groter dan 10 (A < C OR A > 10).
Onderstaande tabel geeft voor de NOT het verband tussen de operand en het resultaat. De NOT operator heeft maar één operand en die volgt direct op de NOT.

A	NOT A
true	false
false	true

De operatoren AND en OR hebben twee operanden. De operator staat bij AND en OR tussen twee operanden in.

A	B	A AND B	A OR B
true	true	true	true
false	false	false	false
true	false	false	true
false	true	false	true

Het uitwerken van logische expressies gaat weer volgens zogenaamde *prioriteitsregels.* Als we deze regels combineren met de regels voor rekenkundige en relationele operatoren, krijgen we de volgende prioriteiten, die in de meeste personal computers gelden:

Prioriteit	Operaties
Hoog ↑	Expressies tussen haakjes
	Machtsverheffen
	Vermenigvuldigen en delen
	NOT
	Optellen en aftrekken
	Relationele operatoren
	AND
Laag	OR

Bij het gebruik van de verschillende soorten operatoren moeten we er rekening mee houden dat de operanden, waarop de operatoren werken, elke keer van de juiste soort (type) zijn.

Operator	Operand(en)	Resultaat
rekenkundig	numeriek	numeriek
relationeel	numeriek	logisch
logisch	logisch	logisch

Voorbeelden

Gecombineerde expressie: X < = Y AND A > B
met: A = 3, B = 2, X = 2 en Y = 5

X < Y AND A > B	
2 < 5 AND 3 > 2	Substitutie (invullen) van de variabelen.
T AND 3 > 2	Linkse relationele expressie (T = true).
T AND T	Rechtse relationele expressie.
T	AND operatie.

Gecombineerde expressie: X < Y AND A > B OR NOT (A < 3) met: A = 4, B = 5, X = 2 en Y = 5	
X < Y AND A > B OR NOT (A < 3)	
2 < 5 AND 4 > 5 OR NOT (4 < 3)	Substitutie van de variabelen.
2 < 5 AND 4 > 5 OR NOT F	Expressie tussen haakjes. (F = false)
2 < 5 AND 4 > 5 OR T	NOT operatie
T AND 4 > 5 OR T	Linkse relationele expressie.
T AND F OR T	Rechtse relationele expressie.
F OR T	AND operatie.
T	OR operatie.

4.5. Voorbeelden met het IF. THEN statement

Na het uitwerken van alle mogelijkheden met logische en relationele operatoren, kunnen we nu dieper ingaan op de conditionele sprongstatements. Dit doen we aan de hand van enkele voorbeelden. Het eerste voorbeeld vinden we in het programma van afbeelding 4.5. We zien in regel 30 een IF. THEN statement. De test, die hierin wordt uitgevoerd, is: kijk of de waarde van de variabele A groter of gelijk is aan 10. Als dit waar is, zal de sprong moeten worden uitgevoerd.
De rest van dit programma is vrij eenvoudig van opzet. De functie is dat een variabele A, die in het begin de waarde 1 krijgt via een lus (regels 40 en 50) telkens met 1 verhoogd wordt.
Afhankelijk van de conditie in regel 30 zal verder worden gegaan met de regels 35, 40, 50 of de regels 90, 100. Als A kleiner is dan 10, kiest de computer het eerste pad. Hierbij wordt dan een tekst afgedrukt die in regel 35 wordt gedefinieerd. De tekst die in dit voorbeeld is gekozen, zegt welke waarde A heeft ten opzichte van 10 en of de conditie true of false is. Bedenk dat het bovenstaande programma geen enkel nut zal hebben, behalve dat het de werking van een conditioneel statement illustreert.
Een toepassing die wel zinvol zou kunnen zijn, is de volgende: we moeten ons voorstellen dat we een programma moeten maken om de *druk* van een of ander ketelhuis te bewaken. Als de druk te hoog wordt, moet er een *alarm* worden gegeven. Tevens moeten de ingevoerde waarden op *fouten* worden getest. Stel dat de variabele D de druk voorstelt. We zullen in het voorbeeld de druk via een INput statement invoeren. Dit had ook anders kunnen gebeuren. Als de druk lager dan 1 is hebben we een *invoerfout* en vraagt de computer

```
*LIST
10      LET A=1.0
20      PRINT
25      PRINT "A=";A
30      IF A>=10.0 THEN 90
35      PRINT A;" >= 10: CØNDITIE IS FALSE"
40      LET A=A+1.0
50      GØTØ 20
90      PRINT A;" >= 10: CØNDITIE IS TRUE"
100     END
*RUN

A= 1.0
 1.0 >= 10: CØNDITIE IS FALSE

A= 2.0
 2.0 >= 10: CØNDITIE IS FALSE

A= 3.0
 3.0 >= 10: CØNDITIE IS FALSE

A= 4.0
 4.0 >= 10: CØNDITIE IS FALSE

A= 5.0
 5.0 >= 10: CØNDITIE IS FALSE
```

(vervolg van de output)

```
A= 6.0
 6.0 >= 10: CØNDITIE IS FALSE

A= 7.0
 7.0 >= 10: CØNDITIE IS FALSE

A= 8.0
 8.0 >= 10: CØNDITIE IS FALSE

A= 9.0
 9.0 >= 10: CØNDITIE IS FALSE

A= 10.0
 10.0 >= 10: CØNDITIE IS TRUE
END PRØGRAM
*
```

Afb. 4.5. Programma me conditietest

om nieuwe data. Wanneer de druk tussen 1 en 50 ligt, is de druk normaal. Van 50 tot 75 nadert de druk een kritische grens, vanaf 75 is de druk kritiek.
Afbeelding 4.6 laat het programma zien. In de regels 22, 30, 40 en 50 wordt de druk vergeleken met de opgegeven grenswaarden. Afhankelijk van het resultaat springen we naar een regel in het programma waar de bijbehorende boodschap wordt afgedrukt. We keren telkens terug naar regel 20 via het GOTO statement.

```
*LIST
10      REM CØNTRØLE VAN DRUK
20      INPUT "VØER DRUK IN";D
21      PRINT
22      IF D<1.0 THEN 90
30      IF D>=1.0 AND D<50.0 THEN 100
40      IF D>=50.0 AND D<75.0 THEN 110
50      IF D>=75.0 THEN 120
90      PRINT "INVØERFØUT!"
95      GØTØ 20
100     PRINT "DRUK NØRMAAL."
105     GØTØ 20
110     PRINT "DRUK BIJNA KRITIEK!!!"
115     GØTØ 20
```

Afb 4.6. Grensbewaking met behulp van een BASIC programma

```
120    PRINT "*********************"
121    PRINT "*      ALARM        *"
122    PRINT "*                   *"
123    PRINT "* DRUK IS KRITIEK   *"
124    PRINT "*********************"
125    GØTØ 20
130    END
*RUN
VØER DRUK IN?-7
INVØERFØUT!
VØER DRUK IN?30
DRUK NØRMAAL.
VØER DRUK IN?60
DRUK BIJNA KRITIEK!!!
VØER DRUK IN?90
*********************
*      ALARM        *
*                   *
* DRUK IS KRITIEK   *
*********************
VØER DRUK IN?!

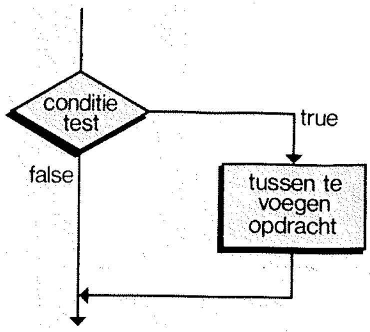
BREAK IN LINE 20
*
```

Naast het IF. THEN statement heeft BASIC ook nog een andere verschijningsvorm voor een conditionele sprongopdracht, namelijk het IF. GOTO statement.
Deze werkt precies hetzelfde als het hiervoor beschreven IF. THEN statement. Afbeelding 4.7 laat de aanpassingen in het programma

```
*LIST
10     REM CØNTRØLE VAN DRUK
20     INPUT "VØER DRUK IN";D
21     PRINT
22     IF D<1.0 GØTØ 90
30     IF D>=1.0 AND D<50.0 GØTØ 100
40     IF D>=50.0 AND D<75.0 GØTØ 110
50     IF D>=75.0 GØTØ 120
90     PRINT "INVØERFØUT!"
95     GØTØ 20
100    PRINT "DRUK NØRMAAL."
105    GØTØ 20
110    PRINT "DRUK BIJNA KRITIEK!!!"
115    GØTØ 20
120    PRINT "*********************"
121    PRINT "*      ALARM        *"
122    PRINT "*                   *"
123    PRINT "* DRUK IS KRITIEK   *"
124    PRINT "*********************"
125    GØTØ 20
130    END
*
```

Afb. 4.7. Programma met de IF. GOTO constructie

zien. Veel versies van BASIC hebben nog een uitbreiding van het IF. THEN statement. Het komt namelijk in de praktijk vaak voor dat we niet willen springen als een bepaalde conditie *true* is. We willen dan een of andere handeling tussenvoegen.
In het flow-diagram van afbeelding 4.8 wordt door de computer het pad met de tekst 'tussen te voegen opdracht' uitgevoerd indien de conditie *true* is. Als de conditie *false* is, wordt de opdracht overgeslagen.
Als we deze constructie in BASIC willen uitvoeren, moeten we achter het keyword THEN niet een regelnummer plaatsen, maar een normaal BASIC statement (echter *zonder* regelnummer). Later zal blijken dat we zelfs een reeks opdrachten kunnen tussenvoegen als we gebruik maken van een GOSUB statement achter de THEN.

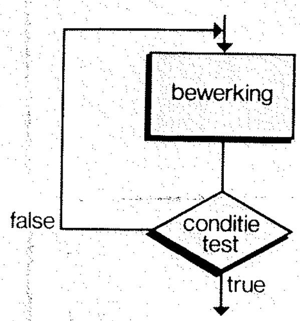


Afb 4.8. Tussenvoegen van een programmadeel door middel van de IF. THEN constructie



Rekencentrum TH Eindhoven

Een veel gebruikte toepassing van de IF. THEN constructie vinden we in programma's waarin net zo lang een herhaling van een deel van het programma moet plaatsvinden totdat een conditie is veranderd. In afbeelding 4.9 vinden we hiervan een flow-diagram. In het blok met de naam 'bewerking' zullen op de een of andere wijze de argumenten van de testen conditie moeten kunnen veranderen. Als dit niet het geval is, hebben we een oneindige lus gemaakt. Het programma *hangt zichzelf op* zoals men in de computerwereld zegt. Het veranderen van de betreffende conditie kan het gevolg zijn van een berekening of van nieuwe inputgegevens. Een voorbeeld dat het nut van de besproken constructie laat zien, is het volgende: wanneer we een programma hebben gemaakt, zou de computer aan de gebruiker kunnen vragen of het programma nogmaals moet worden uitgevoerd, of dat er moet worden gestopt. Als we een 1 invoeren zou dit betekenen: *herhalen*. Een ander getal betekent: *stop ermee*.



Afb. 4.9. Programmalus met behulp van het IF. THEN statement

Het voorbeeldprogramma moet, nadat de koers van een dollar is ingevoerd, een omzetting van een bedrag in dollars naar guldens realiseren (zie voor het programma afbeelding 4.10).

```
*LIST
10     PRINT "KØERSØMZETTING"
20     PRINT
30     INPUT "$1.00 IS IN GULDEN";A
35     PRINT
40     INPUT "BEDRAG IN $";B
50     PRINT
60     PRINT "$";B;" IS FL";B*A
70     PRINT "WILT U HET PRØGRAMMA HERHALEN?"
80     INPUT "VØER DAN 1 IN";C
90     IF C=1.0 THEN 35
95     END
*RUN
KØERSØMZETTING

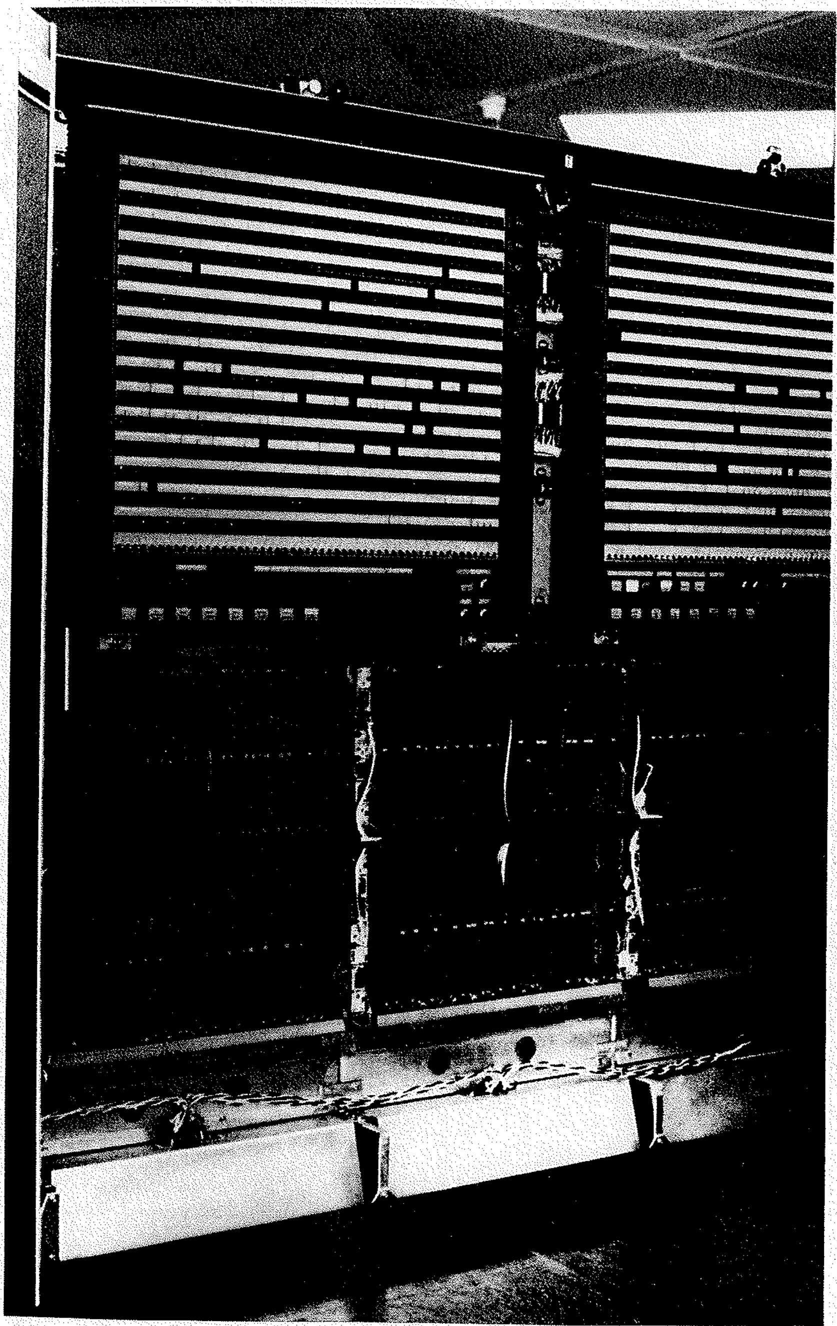
$1.00 IS IN GULDEN?1.98
BEDRAG IN $?2.34
$ 2.34 IS FL 4.6332
WILT U HET PRØGRAMMA HERHALEN?
VØER DAN 1 IN?1
BEDRAG IN $?12.56
$ 12.56 IS FL 24.8688
WILT U HET PRØGRAMMA HERHALEN?
VØER DAN 1 IN?0END PRØGRAM
*
```

Afb. 4.10. Conditioneel herhalen van een programma

```
*LIST
10     LET A=1.0
20     IF A>10.0 THEN 90
30     PRINT "A =";A
40     LET A=A+1.0
50     REM -----------------------
55     REM
60     REM EEN ØF ANDERE BEWERKING
65     REM
70     REM -----------------------
80     GØTØ 20
90     END
*RUN
A = 1.0
A = 2.0
A = 3.0
A = 4.0
A = 5.0
A = 6.0
A = 7.0
A = 8.0
A = 9.0
A = 10.0
END PRØGRAM
*
```

Afb. 4.11. Programmalussen

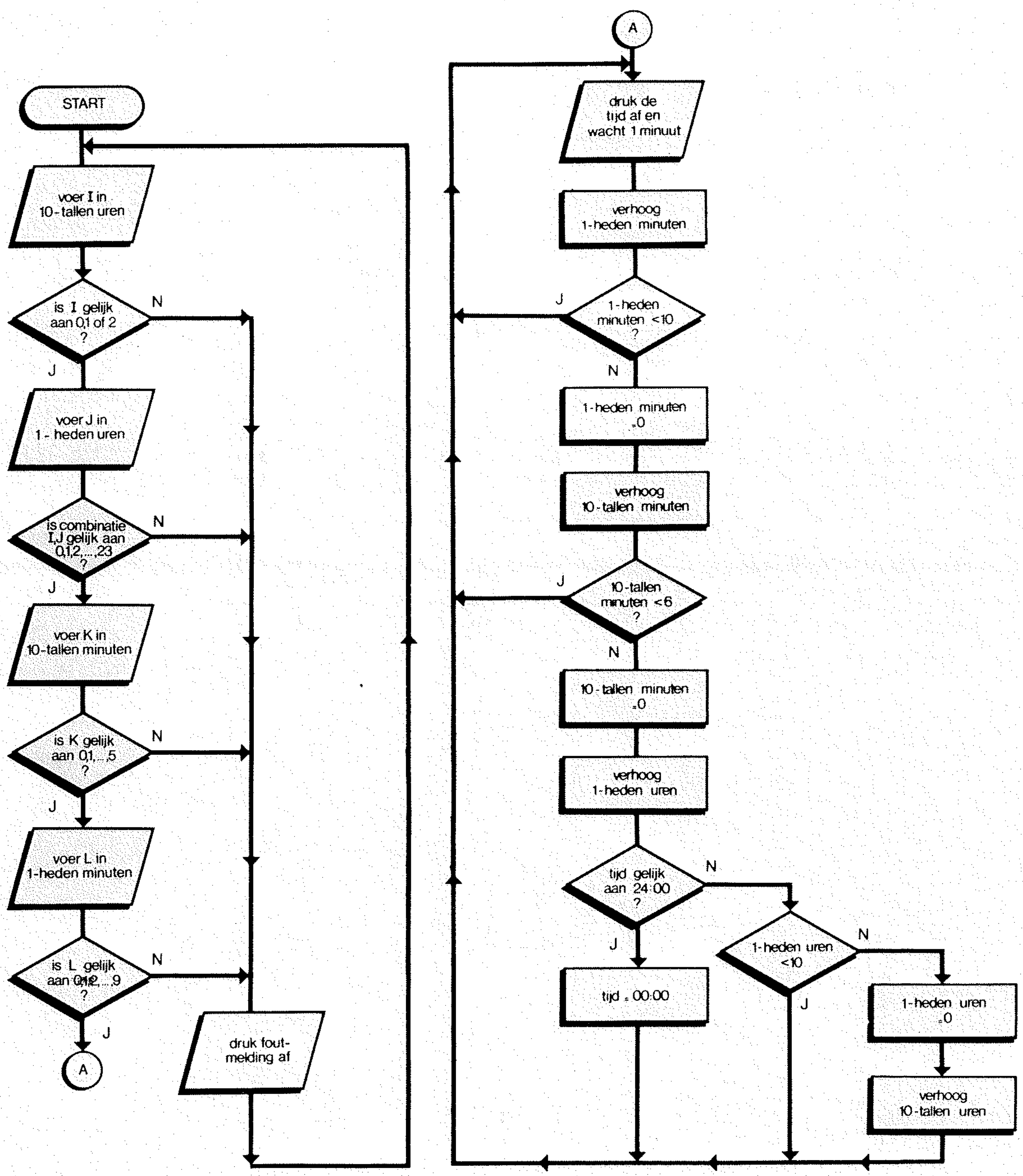
Een ander veel voorkomend gebruik van het IF. THEN statement zien we terug in het programma van afbeelding 4.11. Hierbij willen we 'een of andere bewerking', die we willekeurig kunnen invullen, een vast aantal laten uitvoeren. Dit realiseren we met de variabele A die we van een *initiële* (begin) waarde telkens met 1 verhogen nadat we de bewerking hebben uitgevoerd. We springen dan terug in het programma en testen of een grenswaarde is overschreden. Afhankelijk van de test voeren we het bewerkingsblok nogmaals uit, verhogen de variabele weer, testen weer, enz. (Deze constructie komen we later nog tegen in de vorm van een FOR/NEXT constructie.)



Wanneer wordt deze computer tot één chip.

Met behulp van de IF. THEN instructie zullen we een programma maken dat de computer als *digitale klok* laat functioneren. In het begin van het programma voeren we de *momentane* tijd in. Eerst de tientallen van de uren (I), dan de eenheden van de uren (J), daarna de tientallen van de minuten (K) en als laatste de eenheden van de minuten (L). Tijdens de invoerfase controleren we op eventuele invoerfouten De tientallen van de uren mogen 0, 1 of 2 zijn.
De eenheden van de uren mogen 0 t/m 9 zijn als de tientallen van de uren 0 of 1 zijn. Zijn de tientallen van de uren 2 dan mogen de eenheden 0, 1, 2 of 3 zijn. Ga eens na hoe dit deel van het programma

Afb. 4.12. Stroomschema van een klokprogramma



werkt. Doe dit aan de hand van het *stroomschema* van afbeelding 4.12 en de *programmalisting* van afbeelding 4.13. In het programma wordt de BASIC functie INT gebruikt. Hierdoor wordt de gehele waarde genomen van het getal dat tussen de haakjes staat. Later zal dit nader worden uitgewerkt.

Afb. 4.13. Een klokprogramma in BASIC

```
*LIST
100   PRINT "VØER DE TIJD IN."
110   PRINT
120   INPUT "10-TALLEN VAN DE UREN";I
125   PRINT
130   IF I<>INT(I) ØR I<0.0 ØR I>2.0 THEN 900
135   INPUT "1-HEDEN VAN DE UREN";J
136   PRINT
140   IF J<>INT(J) ØR J<0.0 ØR J>9.0 ØR (J>3.0 AND I=2.0) THEN 900
145   INPUT "10-TALLEN VAN DE MINUTEN";K
146   PRINT
150   IF K<>INT(K) ØR K<0.0 ØR K>5.0 THEN 900
155   INPUT "1-HEDEN VAN DE MINUTEN";L
156   PRINT
160   IF L<>INT(L) ØR L<0.0 ØR L>9.0 THEN 900
162   PRINT :PRINT :PRINT :MØDE 0
163   PRINT "DE TIJD IS:"
164   PRINT
165   LET I%=I
166   LET J%=J
167   LET K%=K
168   LET L%=L
169   PRINT I%;J%;" : ";K%;L%
210   LET M=1.0
215   LET M=M+1.0
220   IF M<10000.0 THEN 215
230   LET L=L+1.0
240   IF L<10.0 THEN GØTØ 162
250   LET L=0.0
260   LET K=K+1.0
270   IF K<6.0 GØTØ 162
280   LET K=0.0
290   LET J=J+1.0
300   IF I=2.0 AND J=4.0 GØTØ 600
310   IF J<10.0 GØTØ 162
320   LET J=0.0
330   LET I=I+1.0
340   GØTØ 162
600   LET I=0.0
610   LET J=0.0
620   LET K=0.0
630   LET L=0.0
640   GØTØ 162
900   REM INVØER FØUT GEDETECTEERD
910   PRINT "INVØERFØUT!"
920   GØTØ 100
999   END
```

(uitvoering van het programma)

```
*RUN
VØER DE TIJD IN.

10-TALLEN VAN DE UREN?2
1-HEDEN VAN DE UREN?1
10-TALLEN VAN DE MINUTEN?6
INVØERFØUT!
VØER DE TIJD IN.

10-TALLEN VAN DE UREN?2
1-HEDEN VAN DE UREN?1
10-TALLEN VAN DE MINUTEN?1
1-HEDEN VAN DE MINUTEN?8

DE TIJD IS:

 2 1 :  1 8



DE TIJD IS:

 2 1 :  1 9



DE TIJD IS:

 2 1 :  2 0
```

Als de invoer goed is, wordt de tijd afgedrukt. Daarna voert het programma een lus uit die precies 1 minuut moet duren. Deze tijd moeten we proefondervindelijk vaststellen. Een lus, die 10 000 keer wordt doorlopen, klopt globaal voor de Teleac/DAI personal computer.
In de rest van het programma wordt de tijd met 1 verhoogd. We moeten hierbij letten op de overgang van 59 minuten naar een uur en op de overgang van 23:59 naar 0 uur.
De klok zal door het verschil in luslengte bij de diverse doorgangen enige onnauwkeurigheid bevatten.

Statement	Effect
IF. THEN	Conditionele executie of conditionele sprong

Algemene vorm:
regelnummer IF x THEN s

> waarbij x een relationele expressie is en s een BASIC statement of een regelnummer.

Effect:
In het IF. THEN statement wordt een *relationele* expressie uitgewerkt. De waarde van de expressie stelt een conditie voor, die getest wordt. Als de waarde van de conditie 'true' is, zal worden verder gegaan met het statement, dat direct volgt op het keyword THEN. Indien hier geen statement achter THEN staat, maar een regelnummer, zal een sprong naar de betreffende regel worden uitgevoerd.
Wanneer de conditie de waarde 'false' heeft, zal worden verder gegaan met het BASIC statement op de regel, die direct volgt op het IF. THEN statement.

Statement	Effect
IF. GOTO	Conditionele sprong

Algemene vorm:
regelnummer IF x GOTO n

> waarbij x een relationele expressie is en n een regelnummer.

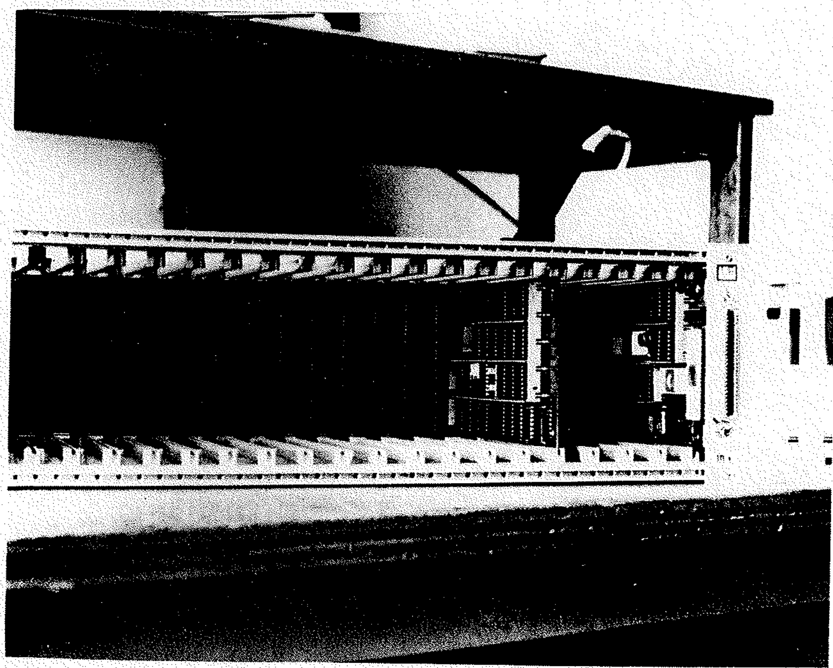
Effect:
In het IF. GOTO statement wordt de waarde van de relationele expressie bepaald. Dit betekent dat er een conditie wordt *getest.* Als de conditie de waarde 'true' heeft, wordt er een sprong uitgevoerd naar het regelnummer n dat volgt op het keyword GOTO. Indien de conditie de waarde 'false' heeft, wordt niet gesprongen, maar verder gegaan op de regel, die direct volgt op de regel met het IF. GOTO statement.

4.6. Foutdiagnose

Een programma zal de eerste keer meestal niet helemaal correct werken. De oorzaak ligt in de fouten, die de programmeur of degene, die het probleem heeft geanalyseerd, heeft gemaakt. De bron van fouten is velerlei. Veel voorkomende oorzaken zijn:
- het probleem is onvoldoende gedefinieerd;
- er zijn denkfouten gemaakt in de oplossing;
- er zijn programmeerfouten gemaakt;
- het programma is niet correct ingetikt.

Bij gebruiken van een RUN commando zal blijken dat een aantal van de bovengenoemde fouten zichzelf presenteren. Dit zijn de fouten die zijn gemaakt tegen de statements van de taal BASIC. Voorbeelden van deze grammaticale of syntactische fouten zijn:
- foute statements;
- ontbreken van statementnummers;
- foute haakjesparen;
- verkeerde typen variabelen in expressies;
- ontbreken van het END statement.

Wanneer de computer een BASIC statement, waarin een *syntactische* fout zit, moet uitvoeren, zal een foutmelding worden afgedrukt. Deze boodschap zal in het kort iets vertellen over het soort fout dat is opgetreden en de plaats waar de fout is gevonden.
Voorbeelden hiervan zijn:
- SYNTAX ERROR;
- COMMAND INVALID;
- TYPE MISMATCH;
- NEXT WITHOUT FOR;
- UNDEFINED LINENUMBER.



De microcomputer (CPU) rechts kan een totaal systeem van 20 kaarten (RAM, I/O e.d.) controleren (8080 microcomputersysteem Vakgroep Digitale Systemen, TH Eindhoven).

De fabrieksdocumentatie zal een uitleg geven van de betekenis van deze foutmeldingen.
De bovengenoemde fouten zijn meestal erg eenvoudig op te sporen en te herstellen (zie hiervoor ook het deel dat beschrijft hoe *tikfouten* worden verbeterd, regels worden toegevoegd of weggelaten). Pas op voor de letter O en het cijfer 0 (nul). Ze worden vaak verwisseld. Ook zijn er vaak printers die diverse vormen hebben voor een character. Zo is nul soms: 0 of Ø en het teken voor machtsverheffen ∧ of ↑. Let op want in de tekst is opzettelijk gewisseld.
Een echt probleem vormen echter de logische fouten. Hiermee bedoelen we denk- en analysefouten. Vaak zal blijken dat het effect van een ontwikkeld programma anders is dan we verwachten. De computer voert het ingevoerde programma wel correct uit, maar de oorzaak van de fout moeten we bij onszelf zoeken. Meestal hebben we niet goed genoeg nagedacht. We hebben nu namelijk een ander programma gemaakt dan de bedoeling was.
Het zou niet van realiteitszin getuigen om te zeggen dat men geen fouten zal maken bij het schrijven van computerprogramma's. Beter gezegd: bijna elk programma is de eerste keer fout.
Voor het opsporen van programmeerfouten geldt dat het aantal fouten snel minder wordt als men een *structuur* in de programma's brengt. Dit betekent dat de opbouw van de programma's dezelfde vorm zal moeten krijgen als een flowdiagram. Later zal hierop dieper worden ingegaan.
Het probleem waar we nu nog mee zitten is: hoe kan een beginnende programmeur zijn fouten opsporen en verbeteren? Het verbeteren van fouten kan zowel het maken van een nieuwe systeemstudie betekenen als het corrigeren van kleine delen van het programma.
Het opsporen van fouten is echter een veel moeilijker techniek. Het zal vaak zinvol zijn om een programma te testen met een inputcombinatie waarvan de resultaten eenvoudig zijn te bepalen. We kunnen dan direct zien of het programma voor deze gegevens (en misschien ook voor andere) correct werkt. Is dit niet het geval, dan kunnen we als volgt te werk gaan:

1. Maak een studie van de blokken statements, die een logische eenheid vormen. We moeten hiervan precies weten welke functies er moeten worden uitgevoerd en wat de resultaten ervan zijn. Dit kunnen we dan vergelijken met het programma dat we meenden geschreven te hebben. Het opdelen van het probleem hebben we ook al gedaan bij het flow-diagram. Het programma moet hiermee een *eenduidige* relatie hebben.
2. Als na het bestuderen van het programma de fout niet is achterhaald, kunnen we het programma een tussentijdse output laten verzorgen. We voegen dan aan het programma op een aantal essentiële punten PRINT statements toe. Deze punten kunnen bijvoorbeeld de plaatsen zijn waar we een *subroutine* aanroepen of een belangrijke berekening hebben uitgevoerd. We kunnen op deze plaatsen de waarde van de diverse variabelen laten afdrukken en vergelijken met wat we ervan hadden verwacht. Ook is het vaak zinvol om bij IF. THEN, ON. GOTO en ON. GOSUB statements te onderzoeken in welke richting het programma verder gaat. In de praktijk worden namelijk tegen de conditietests veel fouten gemaakt. ON. GOTO en ON. GOSUB komen later nader aan de orde.

Het is zinvol om bij het toevoegen van PRINT statements van voren naar achteren te werken, omdat we dan telkens kunnen voortbouwen op reeds geteste delen van het programma. De fout zal worden gevonden als we telkens het resultaat toetsen aan de verwachtingen.

3. Wanneer echter op deze manier de fout niet te lokaliseren is, zullen we het programma nu niet meer in zijn geheel aan kunnen pakken, maar in losse delen moeten testen. Hiervoor bestaat de mogelijkheid om via een STOP statement de executie af te breken. We kunnen dan via een direct statement de waarde van de variabele laten afdrukken of veranderen. Een directe executie houdt in dat we een statement intikken zonder regelnummers. BASIC zal dan direct de opdracht uitvoeren. We kunnen zo op elke plaats in het programma stoppen. Daarna kan door middel van een CONT (continue) opdracht op de plaats worden verder gegaan waar we zojuist de executie hebben afgebroken. Zo kan op elke plaats een input worden ingevoerd.
4. Soms weten we niet precies wat de computer doet of waar we bezig zijn. We kunnen nu via de BREAK toets de executie stoppen en aan het door de computer afgedrukte regelnummer zien op welke plaats we het programma hebben afgebroken. Daarna staan ons weer alle zojuist besproken *hulpmiddelen* ter beschikking.
5. Als al deze methodes niet werken, moeten we zelf voor computer spelen en stap voor stap noteren wat er moet gebeuren en daarna de computer de stap uit laten voeren. Door nu de notities te vergelijken met de resultaten zullen erg hardnekkige fouten uit het programma te voorschijn komen.
6. Als de fouten hiermee nog niet zijn opgespoord, kunnen we ook nog een vergissing hebben gemaakt doordat het programma is opgezet voor een andere set van invoerdata dan die we er aan presenteren.

Het zal duidelijk zijn dat deze manier van fouten zoeken geenszins compleet of waterdicht is, maar toch zullen veel fouten op deze manier uit de programma's naar voren komen.
Bedenk wel dat u meestal zelf de oorzaak van de fouten bent. In afbeelding 4.14 is een voorbeeld gegeven waarin het zoeken naar fouten in een programma stap voor stap is gedemonstreerd.
In de Teleac/DAI personal computer zijn nog enkele extra mogelijkheden aanwezig voor het testen van programma's. Zodra we met behulp van het STOP statement de executie van het programma hebben afgebroken, is het mogelijk om een STEP commando te geven. Hierna zal de computer bij aanslag van de RETURN toets telkens één regel ten uitvoer brengen en laten zien wat er gebeurt. We gaan zo in zogenaamde single step (stap voor stap) door het programma. Willen we doorgaan op volle snelheid dan moeten we het commando CONT (continue) geven.
Een andere manier van zichtbaar maken (trace = spoor(zoeken)) van tussentijdse resultaten kan m.b.v. het statement TRON (trace on) en TROFF (trace off). Van alle statements die tussen deze twee liggen wordt net als bij de STEP getoond wat er gebeurt.
Bij de STEP hebben we zelf de snelheid in de hand en bij de trace bepaalt de computer de snelheid.

```
*LIST
10      X=1.0
20      IF X>10.0 THEN 999
25      Y=X↑0.5
30      PRINT "X IS GELIJK AAN : ",X
40      PRINT "DE WØRTEL UIT X IS : ",Y
50      X=X+1.0
60      GØTØ 10
999     END
*RUN
X IS GELIJK AAN :          1.0
DE WØRTEL UIT X IS :       1.0
X IS GELIJK AAN :          1.0
DE WØRTEL UIT X IS :       1.0
X IS GELIJK AAN :          1.0
BREAK IN LINE 30
*55 PRINT X
*LIST
10      X=1.0
20      IF X>10.0 THEN 999
25      Y=X↑0.5
30      PRINT "X IS GELIJK AAN : ",X
40      PRINT "DE WØRTEL UIT X IS : ",Y
50      X=X+1.0
55      PRINT X
60      GØTØ 10
999     END
*RUN
X IS GELIJK AAN :          1.0
DE WØRTEL UIT X IS :       1.0
 2.0
X IS GELIJK AAN :          1.0
DE WØRTEL UIT X IS :       1.0
 2.0
X IS GELIJK AAN :          1.0

BREAK IN LINE 30
*26 PRINT X
*LIST
10      X=1.0
20      IF X>10.0 THEN 999
25      Y=X↑0.5
26      PRINT X
30      PRINT "X IS GELIJK AAN : ",X
40      PRINT "DE WØRTEL UIT X IS : ",Y
50      X=X+1.0
55      PRINT X
60      GØTØ 10
999     END
*RUN
 1.0
X IS GELIJK AAN :          1.0
DE WØRTEL UIT X IS :       1.0
```

(x verandert niet)

BREAK

extra printopdracht

x wordt wel verhoogd

x wordt wel verhoogd

BREAK

extra printopdracht

Afb. 4.14. Foutdiagnose

(vervolg van het programma)

```
 2.0
 1.0
X IS GELIJK AAN :           1.0
DE WØRTEL UIT X IS :        1.0
 2.0
 1.0
X IS GELIJK AAN :           1.0
BREAK IN LINE 30
*60 GØTØ 20
*LIST
10      X=1.0
20      IF X>10.0 THEN 999
25      Y=X↑0.5
26      PRINT X
30      PRINT "X IS GELIJK AAN : ",X
40      PRINT "DE WØRTEL UIT X IS : ",Y
50      X=X+1.0
55      PRINT X
60      GØTØ 20
999     END
*RUN
 1.0
X IS GELIJK AAN :           1.0
DE WØRTEL UIT X IS :        1.0
 2.0
 2.0
X IS GELIJK AAN :           2.0
DE WØRTEL UIT X IS :        1.41421
 3.0
 3.0
X IS GELIJK AAN :           3.0
DE WØRTEL UIT X IS :        1.73205
BREAK IN LINE 40
*26
*55
*LIST
10      X=1.0
20      IF X>10.0 THEN 999
25      Y=X↑0.5
30      PRINT "X IS GELIJK AAN : ",X
40      PRINT "DE WØRTEL UIT X IS : ",Y
50      X=X+1.0
60      GØTØ 20
999     END
*RUN
X IS GELIJK AAN :           1.0
DE WØRTEL UIT X IS :        1.0
X IS GELIJK AAN :           2.0
DE WØRTEL UIT X IS :        1.41421
X IS GELIJK AAN :           3.0
DE WØRTEL UIT X IS :        1.73205
X IS GELIJK AAN :           4.0
```

X wordt op 1 teruggezet FOUT!!

BREAK

nu is X wel goed

BREAK

weghalen van regel 26 en 55

(vervolg van het programma)

```
DE WØRTEL UIT X IS :        2.0
X IS GELIJK AAN :           5.0
DE WØRTEL UIT X IS :        2.23607
X IS GELIJK AAN :           6.0
DE WØRTEL UIT X IS :        2.44949
X IS GELIJK AAN :           7.0
DE WØRTEL UIT X IS :        2.64575
X IS GELIJK AAN :           8.0
DE WØRTEL UIT X IS :        2.82843
X IS GELIJK AAN :           9.0
DE WØRTEL UIT X IS :        3.0
X IS GELIJK AAN :           10.0
DE WØRTEL UIT X IS :        3.16228
END PRØGRAM
*
```

Samenvatting hoofdstuk 4

In het eerste deel van dit hoofdstuk is de conditionele sprongopdracht besproken. Hiermee hebben we een gereedschap ter beschikking gekregen, waarmee we de computer echt kunnen gaan inzetten.
In het tweede gedeelte is een aantal methodieken besproken, die van nut kunnen zijn bij het opsporen van logische fouten in de programma's.

Multiple choice-vragen bij hoofdstuk 4

A. Bepaal de waarden van de relationele expressie a en b, als geldt:

A = 2, B = 3, C = 4, en D = 5.

a. A > B OR C < D

b. A + B > C + D OR A/B = C/D

1 expressie a heeft de waarde TRUE en expressie b heeft de waarde TRUE

2 expressie a heeft de waarde FALSE en expressie b heeft de waarde TRUE

3 expressie a heeft de waarde TRUE en expressie b heeft de waarde FALSE

4 expressie a heeft de waarde FALSE en expressie b heeft de waarde FALSE

B. Een conditionele sprong is:

1 een sprong die wordt gemaakt als naar een verder in het programma liggend statement wordt gesprongen

2 een sprong die alleen wordt gemaakt als de bijbehorende conditie de waarde TRUE heeft

3 een sprong die alleen wordt gemaakt als de bijbehorende conditie de waarde FALSE heeft

4 een sprong naar een voorliggend statement

C. Welk statement is fout?

1 10 IF A > B AND P < Q THEN 100

2 10 IF A = 6 AND B = 7 GOTO 100

3 10 IF A < > B AND P = Q THEN GOTO 100

4 10 GOTO 100 IF A < > B AND A = 7

D. **a.** De volgende statements hebben niet hetzelfde effect:
10 IF A > = B THEN 200
en
20 IF A > B THEN IF A = B GOTO 200
b. De volgende statements hebben hetzelfde effect:
10 IF A < 7 THEN 100
en
20 IF A − 7 < 0 THEN 100

1 uitspraak a is goed en uitspraak b is goed
2 uitspraak a is fout en uitspraak b is goed
3 uitspraak a is goed en uitspraak b is fout
4 uitspraak a is fout en uitspraak b is fout

E. **a.** Bij executie van het statement:
10 IF A > B OR A = B OR A < B GOTO 100
springen we nooit naar regel 100.
b. Bij executie van het statement:
10 IF A < B AND A < > B AND A > B GOTO 100
springen we nooit naar regel 100.

1 uitspraak a is goed en uitspraak b is goed
2 uitspraak a is fout en uitspraak b is goed
3 uitspraak a is goed en uitspraak b is fout
4 uitspraak a is fout en uitspraak b is fout

F. **a.** De uitvoering van een programmalus kan worden beëindigd indien een variabele een grenswaarde overschrijdt.
b. Een programmalus kan worden geconstrueerd door middel van de IF. THEN en/of de GOTO statements.

1 uitspraak a is goed en uitspraak b is goed
2 uitspraak a is fout en uitspraak b is goed
3 uitspraak a is goed en uitspraak b is fout
4 uitspraak a is fout en uitspraak b is fout

G. Als in een IF statement de te testen conditie de waarde TRUE heeft:

1 wordt doorgegaan met het statement op de volgende regel
2 wordt doorgegaan met de rest van het IF statement
3 springen we naar een regel
4 wordt de lopende variabele met één verhoogd

H. Programma:

```
10 LET A = 1: LET B = 10
20 LET A = A + B
30 LET B = A: LET A = B
40 IF A < 50 GOTO 30
50 GOTO 20
60 END
```

1 het programma stopt met A en B gelijk aan 11
2 het programma bevat een oneindige lus
3 A en B krijgen om en om de waarde 1 en 10
4 statement 50 wordt bij elke lusdoorgang één keer uitgevoerd

I. Programma:

```
10  LET    S = 1
20  IF     S > 20 THEN 70
30  LET    T = S ↑ 2
40  PRINT S, T
50  LET    S = S + T
60  GOTO 20
70  END
```

Welke waarden hebben S en T als het programma stopt?

1 S = 6 en T = 36
2 S = 42 en T = 36
3 S = 42 en T = 1764
4 S = 1806 en T = 1764

J. Een iteratieve procedure is een procedure, die herhaald wordt uitgevoerd. Dit gebeurt tot een variabele een bepaalde waarde overschrijdt.

Programma:

```
10 LET A = 1: C = 1: LET B = 4
20 LET A = A + B: C = C + 1
30 IF A > = 16 THEN 50
40 GOTO 20
50 PRINT "B = ", B
60 PRINT "C = ", C
```

Hoe vaak wordt deze procedure doorlopen?

1 4 keer
2 5 keer
3 6 keer
4 16 keer

wordt
aard

Hoofdstuk 5

Programmalussen

In dit hoofdstuk komt een van de belangrijkste statements van de taal BASIC naar voren. Door het zogenaamde FOR/NEXT statements wordt het eenvoudig om programma's meerdere malen uit te voeren. Hierdoor kan de computer worden ingezet om routinewerkzaamheden uit te voeren.
In het tweede deel van dit hoofdstuk komen een aantal voor de praktijk nuttige zaken aan de orde. Na het opzetten en het testen van een programma willen we het meestal bewaren om op een later tijdstip weer te kunnen gebruiken. Hiervoor zijn er in BASIC de commando's SAVE en LOAD, die programma's naar een cassetteband wegschrijven of vanaf de cassetteband in het geheugen laden.

5.1. Programmalussen

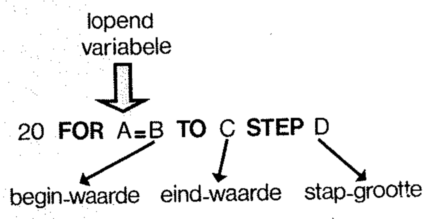
De kracht van de moderne computer ligt in zijn vermogen om zonder het optreden van vermoeidheidsverschijnselen snel reeksen met handelingen te herhalen. In de voorgaande hoofdstukken hebben we gezien hoe deze zogenaamde lussen in een programma kunnen worden geprogrammeerd in de programmeertaal BASIC met behulp van IF. THEN en de GOTO statements. Omdat de *programmalussen* in de praktijk echter vaak voorkomen, is hiervoor in BASIC een speciaal statementpaar gemaakt, namelijk de combinatie:

FOR variabele = beginwaarde TO eindwaarde STEP stapgrootte

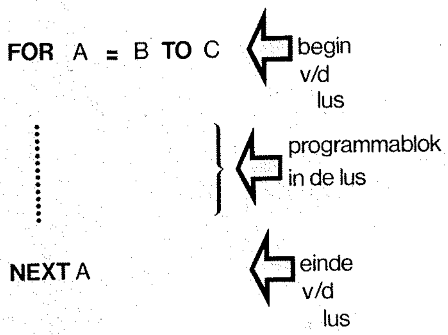
en

NEXT variabele

Tussen de FOR en NEXT statements staat een programmablok dat verschillende malen moet worden uitgevoerd. Het aantal malen wordt door middel van het FOR statement bestuurd.
In de lus is een variabele nodig, waarmee we het aantal malen, dat de lus wordt doorlopen, tellen. Bij deze *hulpvariabele* (de zogenaamde lopende variabele) zal bij elke doorgang van de lus een waarde *(de stapgrootte)* worden opgeteld. De stapgrootte is in de meeste gevallen gelijk aan 1. Het doorlopen van de lus wordt beëindigd als een opgegeven grenswaarde *(de eindwaarde)* wordt overschreden. Indien we alleen maar willen tellen, beginnen we met een beginwaarde 1 en lopen met stappen van 1 naar de eindwaarde.
De eerste van de twee statements die altijd bij elkaar horen, is het FOR statement. Hierin wordt de naam van de lopende variabele gedefinieerd, vastgesteld wat de beginwaarde en eindwaarde is en



Afb. 5.1. Opbouw van het FOR statement



Afb. 5.2. Combinatie van het FOR en NEXT statement

eventueel hoe groot de stapgrootte is. Na het FOR statement volgt het programmablok dat verschillende malen moet worden doorlopen. Het einde van dit blok wordt aangegeven door het NEXT statement. Door in het FOR statement en het NEXT statement de lopende variabele op te nemen, is het voor een computer en later ook voor de gebruiker duidelijk waar het begin en einde van het betreffende blok ligt.
Wanneer we met stappen van 1 door de FOR/NEXT combinatie willen lopen, kunnen we het keyword STEP en de stapgrootte weglaten. Afbeeldingen 5.1 en 5.2 geven een beeld van de opbouw van de FOR/NEXT statements.

In de voorbeelden van afbeeldingen 5.3 t/m 5.6 zijn enkele varianten van de FOR/NEXT combinatie gebruikt. Het programmablok, dat we herhaald doorlopen, is een enkel PRINT statement waarin de waarde van de lopende variabele wordt afgedrukt. In het eerste voorbeeld loopt de variabele I van 1 tot en met 10. De stapgrootte was immers 1, want STEP is weggelaten.
In het tweede voorbeeld is de stapgrootte 3. Nu zien we dat I achtereenvolgens de waarden 1, 1 + 3 = 4, 4 + 3 = 7 en 7 + 3 = 10 aanneemt.

```
*LIST
10     FØR I=1.0 TØ 10.0
20     PRINT "I =";I
30     NEXT I
40     END
*RUN
I = 1.0
I = 2.0
I = 3.0
I = 4.0
I = 5.0
I = 6.0
I = 7.0
I = 8.0
I = 9.0
I = 10.0
END PRØGRAM
*
```

Afb 5.3.

```
*LIST
10     FØR I=1.0 TØ 10.0 STEP 3.0
20     PRINT "I =";I
30     NEXT I
40     END
*RUN
I = 1.0
I = 4.0
I = 7.0
I = 10.0
END PRØGRAM
*
```

Afb. 5.4.

```
*LIST
10     FØR I=10.0 TØ -5.0 STEP -2.7
20     PRINT "I =";I
30     NEXT I
40     END
*RUN
I = 10.0
I = 7.3
I = 4.6
I = 1.9
I =-0.8
I =-3.5
END PRØGRAM
*
```

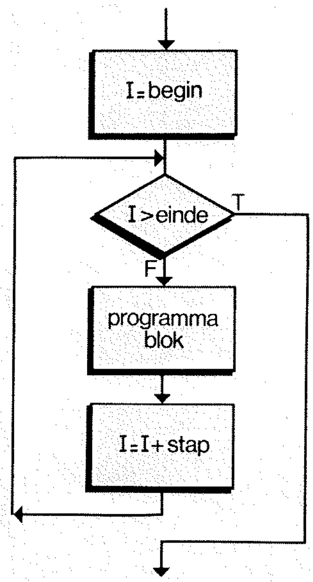
Afb. 5.5.

```
*LIST
10     FØR I=1.0 TØ 10.0 STEP 1.7
20     PRINT "I =";I
30     NEXT I
40     END
*RUN
I = 1.0
I = 2.7
I = 4.4
I = 6.1
I = 7.8
I = 9.5
END PRØGRAM
*
```

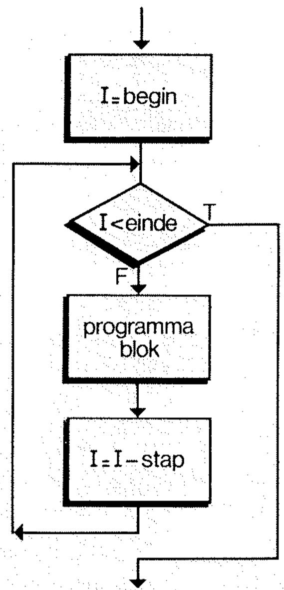
Afb. 5.6.

Afb. 5.3., 5.4., 5.5. en 5.6. Voorbeelden van FOR/NEXT constructies

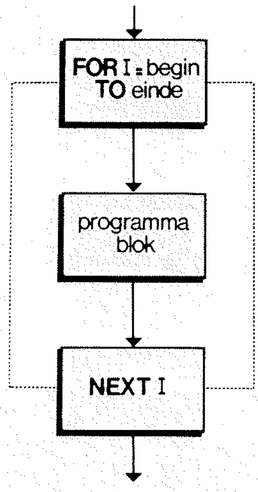
Het derde voorbeeld heeft een stapgrootte die 1.7 is. Het programma spreekt nu voor zichzelf. Het vierde voorbeeld heeft een negatieve stapgrootte, namelijk −2.7. Dit betekent dat de lopende variabele bij elke doorgang van het programmablok met 2.7 zal worden verlaagd. Een tweede verschil, dat vrij groot is, is de grenswaarde −5. Deze waarde is namelijk negatief. In de praktijk betekent dit FOR statement, dat we de lopende variabele niet verhogen, maar verlagen en dat we testen of we voorbij de eindwaarde zijn gegaan door te kijken of de lopende variabele kleiner is dan de grens.



Afb. 5.7. FOR/NEXT structuur met positieve stapgrootte



Afb. 5.8. FOR/NEXT structuur met negatieve stapgrootte



Afb. 5.9. Speciaal symbool voor de FOR/NEXT combinatie

In de flow-diagrammen van afbeeldingen 5.7 en 5.8 wordt nog eens via een schematische weg getoond hoe het FOR statement werkt voor zowel positieve als negatieve grenzen en stapgrootte. Als we van meer complexe programma's een flow-diagram willen maken, is het niet aan te bevelen voor eventuele FOR/NEXT combinaties de zojuist genoemde flow-diagrammen te nemen. Er is voor de FOR/NEXT combinatie een speciaal *symbool* ontworpen, dat in afbeelding 5.9 is weergegeven. In de kop staat het FOR statement. De body wordt gevormd door het programmablok en de staart is het NEXT statement. Via een *stippellijn* wordt het geheel omsloten om te laten zien hoe de statements een eenheid vormen.
Om de bruikbaarheid van de twee besproken statements duidelijk te maken geven we een nieuwe versie van het programma dat dollars omzette in guldens. Bij een studie van het programma uit hoofdstuk 4, afbeelding 4.10, zien we dat de *conversie* in feite gebeurt in het statement op regel 60. Door middel van regels 50, 60, 80 en 90 wordt de lus gerealiseerd. Deze vier regels kunnen we nu vervangen door:

```
50 FOR I = B TO E STEP S
```

en

```
90 NEXT I
```

Het eerste resultaat is een korter programma wat programmatekst betreft, maar een belangrijk tweede gevolg is de betere leesbaarheid. Het programma is veel overzichtelijker geworden en daardoor meer bruikbaar (zie afbeelding 5.10).
Pas op!
Sommige computers gaan altijd één keer door de FOR/NEXT lus, andere niet. Het hangt af van de manier waarop de statements intern zijn gerealiseerd. Sommige computers rekenen in FOR uit hoe vaak ze door de lus gaan. Verandering van de lusvariabele in de lus zelf heeft dan geen effect op het aantal keren dat de lus wordt doorlopen.

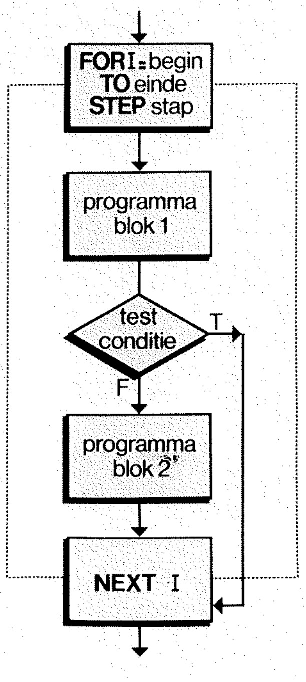
```
*LIST
10    PRINT "KØERSØMZETTING"
15    PRINT "==**==**==**=="
20    PRINT
30    INPUT "$1 IS IN GULDENS";A
31    PRINT
32    INPUT "BEGIN TABEL = $";B
33    PRINT
34    INPUT "EINDE TABEL = $";E
35    PRINT
36    INPUT "STAPGRØØTTE = $";S
37    PRINT
40    PRINT
50    FØR I=B TØ E STEP S
70    PRINT "$";I;" = fl";INT(I*A*100.0+0.5)/100.0
90    NEXT I
100   PRINT
110   PRINT "WILT U HET PRØGRAMMA HERHALEN?"
120   INPUT "VØER DAN 1 IN";C
130   IF C=1.0 THEN 20
140   END
*RUN
KØERSØMZETTING
==**==**==**==
$1 IS IN GULDENS?1.96
BEGIN TABEL = $?1
EINDE TABEL = $?10
STAPGRØØTTE = $?1
$ 1.0 = fl 1.96
$ 2.0 = fl 3.92
$ 3.0 = fl 5.88
$ 4.0 = fl 7.84
$ 5.0 = fl 9.8
$ 6.0 = fl 11.76
$ 7.0 = fl 13.72
$ 8.0 = fl 15.68
$ 9.0 = fl 17.64
$ 10.0 = fl 19.6

WILT U HET PRØGRAMMA HERHALEN?
VØER DAN 1 IN?0END PRØGRAM
*
```

Afb. 5.10. Voorbeeldprogramma met de FOR/NEXT constructie

5.2. Springen uit programmalussen

In veel programma's komt het voor dat we voortijdig uit de lus willen wegspringen omdat we niet verder in het resultaat, dat binnen de lus wordt gegenereerd, zijn geïnteresseerd. In de praktijk betekent dit dat we via een normale IF. THEN test uit de lus springen voordat de lopende variabele de eindwaarde heeft bereikt. In afbeelding 5.11 is in een flow-diagram de besproken constructie van FOR/NEXT met IF. THEN statements grafisch weergegeven. Het programma van afbeelding 5.12 past de constructie toe in een voorbeeld waarbij we in een lus alle kwadraten van gehele getallen willen bepalen. Maar de kwadraten mogen niet boven de waarde 300 uitkomen. In de FOR/NEXT lus laten we de variabele L alle gehele getallen doorlopen zonder acht te slaan op de kwadraten. In de IF. THEN combinatie testen we of we klaar zijn. Is dit het geval dan springen we naar het statement dat direct volgt op het NEXT statement. Hiermee maken we een constructie waarbij we het FOR/NEXT blok maar op één enkele plaats verlaten. Later zal blijken dat dit nuttig is omdat hierdoor de hele programmastructuur duidelijker en ongekunsteld blijft. Het testen en elimineren van fouten gaat nu beter.



Afb. 5.11. Sprong uit een programmalus

```
*LIST
100    REM KWADRATEN ØNDER 300
200    FØR L=1.0 TØ 100.0
300    IF L↑2.0>=300.0 THEN 600
400    PRINT "L = ";L;"  L↑2 = ";L↑2.0
500    NEXT L
600    PRINT "TABEL IS KLAAR."
700    END
*RUN
L =  1.0  L↑2 =  1.0
L =  2.0  L↑2 =  4.0
L =  3.0  L↑2 =  9.0
L =  4.0  L↑2 =  16.0
L =  5.0  L↑2 =  25.0
L =  6.0  L↑2 =  36.0
L =  7.0  L↑2 =  49.0
L =  8.0  L↑2 =  64.0
L =  9.0  L↑2 =  81.0
L =  10.0  L↑2 =  100.0
L =  11.0  L↑2 =  121.0
L =  12.0  L↑2 =  144.0
L =  13.0  L↑2 =  169.0
L =  14.0  L↑2 =  196.0
L =  15.0  L↑2 =  225.0
L =  16.0  L↑2 =  256.0
L =  17.0  L↑2 =  289.0
TABEL IS KLAAR.
END PRØGRAM
*
```

Afb. 5.12. Programma met sprong uit de FOR/NEXT combinatie

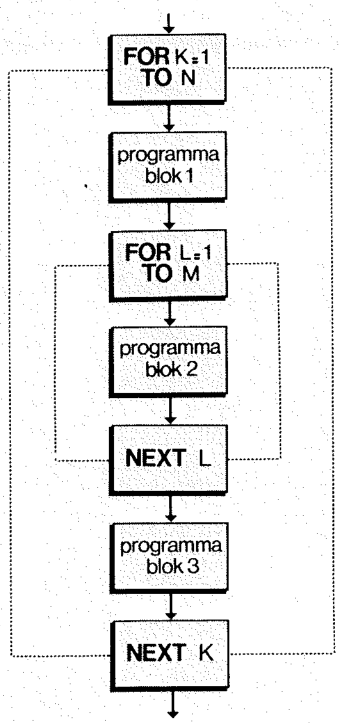
5.3. Lussen binnen lussen

Een andere constructie, die we in veel programma's kunnen gebruiken, bestaat uit de zogenaamde lussen binnen lussen (of: *nested loops*). Hierbij omvat een buitenlus een of meer binnenlussen. Het nut hiervan is dat een programmablok, zoals we in het voorgaande reeds hebben besproken, op zichzelf ook weer uit lussen zal mogen bestaan.

Bij het *nesten van lussen* moeten we de volgende regels in acht nemen:

a. elke lus moet beginnen met een FOR statement en eindigen met een NEXT statement (met hierbij dezelfde lopende variabele);
b. een binnenlus en een buitenlus mogen nooit dezelfde lopende variabele hebben;
c. de binnenlus moet helemaal binnen de buitenlus liggen. Dit wil zeggen dat ze elkaar nooit gedeeltelijk mogen overlappen. De buitenlus omvat de binnenlus helemaal;
d. vanuit de binnenlus mogen we wel naar de buitenlus springen, maar vanuit een buitenlus mag nooit naar de binnenlus worden gesprongen, behalve dan naar het beginpunt van de lus (dus het punt waar het FOR statement staat).
e. het aantal keren dat een lus mag worden aangeroepen is beperkt. De computer moet steeds onthouden waar naar toe terug gegaan wordt.

Bekijk de handleiding van uw computer om na te gaan hoe vaak deze nesting is toegestaan.



Afb. 5.13. Nesting van programmalussen

```
*LIST
100    REM LUSSEN BINNEN LUSSEN
200    FØR I=1.0 TØ 4.0
300    FØR J=1.0 TØ 3.0
400    PRINT "I = ";I;"   J = ";J
500    NEXT J
600    NEXT I
700    END
*RUN
I =  1.0   J =  1.0
I =  1.0   J =  2.0
I =  1.0   J =  3.0
I =  2.0   J =  1.0
I =  2.0   J =  2.0
I =  2.0   J =  3.0
I =  3.0   J =  1.0
I =  3.0   J =  2.0
I =  3.0   J =  3.0
I =  4.0   J =  1.0
I =  4.0   J =  2.0
I =  4.0   J =  3.0
END PRØGRAM
*
```

Afb. 5.14. Programma met een dubbele FOR/NEXT lus

In afbeelding 5.13 is de 'geneste' FOR/NEXT structuur getekend. Het programma van afbeelding 5.14 geeft een voorbeeld van een dubbele lus. Het programmablok van de binnenlus drukt de waarde van de lopende variabele I van de buitenlus en J van de binnenlus af. We zien dat de binnenlus telkens voor elke waarde van de buitenlus geheel wordt doorlopen. Voor I = 1 wordt J achtereenvolgens gelijk aan 1, 2 en 3 en dan wordt I verhoogd en loopt J weer van 1 via 2 naar 3, enz.
Het programma is identiek aan het programma met de IF/GOTO structuren van het programma in afbeelding 5.15.

Een andere toepassing zien we in het programma van afbeelding 5.16. Dit programma bouwt een vierkant (tabel) op van vijf bij vijf elementen. In elk element willen we een getal zetten waaraan we kunnen zien op welke rij en kolom het element ligt. Daartoe nummeren we de rijen in stappen van 10 en de kolommen in stappen van 1. De waarde berekenen we door de waarde van de twee lopende variabelen I en J bij elkaar op te tellen.
Ga zelf na hoe het programma werkt en probeer eens een programma te maken dat de kolommen een gewicht geeft van 10 en de rijen een waarde van 1.

```
*LIST
100   REM LUSSEN BINNEN LUSSEN
150   LET I=1.0
200   IF I>4.0 THEN 600
300   LET J=1.0
350   IF J>3.0 THEN 525
450   PRINT "I = ";I;"   J = ";J
475   LET J=J+1.0
500   GØTØ 350
525   LET I=I+1.0
550   GØTØ 200
600   END
*RUN
I =  1.0   J =  1.0
I =  1.0   J =  2.0
I =  1.0   J =  3.0
I =  2.0   J =  1.0
I =  2.0   J =  2.0
I =  2.0   J =  3.0
I =  3.0   J =  1.0
I =  3.0   J =  2.0
I =  3.0   J =  3.0
I =  4.0   J =  1.0
I =  4.0   J =  2.0
I =  4.0   J =  3.0
END PRØGRAM
*
```

Afb 5.15. Dubbele lus met het IF. THEN statement

```
*LIST
100   FØR I%=10.0 TØ 50.0 STEP 10.0
200   FØR J%=1.0 TØ 5.0
300   PRINT I%+J%;
400   NEXT J%
500   PRINT
600   NEXT I%
700   END
*RUN
 11 12 13 14 15
 21 22 23 24 25
 31 32 33 34 35
 41 42 43 44 45
 51 52 53 54 55
END PRØGRAM
*
```

Afb. 5.16. Toepassing van een dubbele FOR/NEXT structuur

Statement	Effect
FOR. TO	Opzetten van een programmalus

Algemene vorm:
regelnummer FOR v = x_1 TO x_2 [STEP x_3]

waarbij v een numerieke variabele is en x_1, x_2 en x_3 numerieke constanten, numerieke variabelen of rekenkundige expressies zijn.

Effect:
Met behulp van het FOR statement wordt een programmalus opgebouwd. In het FOR statement wordt de numerieke variabele v gebruikt als lopende variabele. Na het FOR statement zal een programmablok volgen met de statements, die in de lus herhaald moeten worden uitgevoerd. Het blok wordt afgesloten met een NEXT v statement. De variabele v is hierbij dezelfde als in het FOR statement.
Het blok tussen FOR en NEXT wordt uitgevoerd voor v = x_1; v = x_1 + x_3; v = x_1 + 2. x_3; enz. Dit gebeurt net zo vaak tot de v grenswaarde x_2 overschrijdt. Als [STEP x_3] weggelaten wordt dan geldt x_3 = 1.
Voordat de lus begint, krijgt v de waarde x_1 en worden de waarden van x_2 en x_3 wat de lus betreft vastgelegd. Telkens wanneer de lus moet worden doorlopen, wordt v met x_2 vergeleken en wel als volgt:

a. als de waarde van x_3 positief is, wordt de lus afgebroken indien geldt $v > x_2$ en verder gegaan met het statement, dat direct volgt op het NEXT v statement;
b. als de waarde van x_3 negatief is wordt de lus afgebroken indien $v < x_2$ en verder gegaan achter het NEXT v statement.

Statement	Effect
NEXT	Beëindigen van een programmalus.

Algemene vorm:
regelnummer NEXT v

waarbij v een numerieke variabele is.

Effect:
Een NEXT statement beëindigt een programmalus. De variabele v in het NEXT statement moet dezelfde zijn als de variabele die gebruikt wordt in het bijbehorende FOR statements. Tijdens het uitvoeren van de lus wordt vanuit het NEXT statement teruggesprongen naar het FOR statement. Als in het FOR statement de programmalus wordt afgebroken omdat de eindvoorwaarde wordt overschreden zal worden doorgegaan met de regel, die direct achter het NEXT statement staat.

5.4. SAVE en LOAD

Het zal duidelijk zijn dat een computer nauwelijks bruikbaar is wanneer we elke keer opnieuw het programma helemaal moeten intikken voordat we kunnen overgaan op de executie ervan. De beste oplossing ervoor zou zijn dat we een programma, wanneer dit klaar is, op een of ander *medium* zouden kunnen opslaan zodat later het programma weer kan worden ingelezen. In de wereld van de personal computers is hiervoor de *audiocassetterecorder* gekozen, omdat deze goedkoop en vrij algemeen verspreid is.
Het wegschrijven van een programma, dat in het geheugen van de computer staat, gebeurt door middel van het commando SAVE. Wanneer dit commando is ingetikt, zal op de *systeemterminal* de boodschap verschijnen de cassetterecorder voor opname te starten. (Meestal moeten we dan zowel de opname- als de startknop indrukken.)
Als dit is gebeurd, geven we dit door aan de computer door de RETURN toets in te drukken. De computer weet nu dat de recorder loopt. De data zal nu vanuit de computer naar de cassette worden weggeschreven. Als de computer hiermee klaar is, kunnen we met een gerust hart de spanning uitschakelen.
Als we later weer verder willen gaan met het programma, kan het weer vanaf de cassette in het geheugen van de computer worden geladen door middel van het LOAD commando.

```
*LIST
10    FØR I%=1 TØ 5
20    PRINT I%,I%*I%
30    NEXT I%
40    END
*SAVE "SCHRIJFTEST"

SET RECØRD,START TAPE,TYPE SPACE <—(opnemen)
*CHECK
 ØK  <——————  BREAK om CHECK te stoppen

***BREAK
*NEW
*LIST
*LØAD "SCHRIJFTEST" <————(afspelen)

*LIST
10    FØR I%=1 TØ 5
20    PRINT I%,I%*I%
30    NEXT I%
40    END
*RUN
 1          1
 2          4
 3          9
 4          16
 5          25
END PRØGRAM
*
```

Afb. 5.17. Voorbeelden van het gebruik van een cassetterecorder voor het opslaan van programma's

Na het intikken van het LOAD commando moeten we de recorder afspelen (na eerst terug te hebben gespoeld) en de data wordt weer in het geheugen gezet. Bij veel computers is het mogelijk om het wegschrijven van de programma's onder een bepaalde naam te doen. We kunnen later het betreffende programma dan weer onder dezelfde naam aanroepen bij het inlezen.
In afbeelding 5.17 zien we enkele voorbeelden van het wegschrijven van een programma. Daarna wordt het geheugen van de computer door middel van het NEW commando gewist en 'geLIST'. We zien nu dat het geheugen leeg is.
Na het LOAD commando wordt het onjuist weggeschreven programma weer geladen. We zijn nu weer terug in de *uitgangstoestand*. Het is zinvol om voor het wegschrijven van het programma via een *microfoon* in te spreken welk programma er op de band wordt gezet. Dit maakt een gemakkelijke *controle* mogelijk. (Het wegschrijven van de waarden van variabelen komt aan de orde.)
Om te controleren of het wegschrijven goed is gegaan, heeft de Teleac/DAI computer een CHECK commando. Door middel van het CHECK commando testen we op fouten. Als het opnemen goed is gegaan verschijnt op het scherm de naam van het opgenomen stuk gevolgd door OK. Is er echter een fout gevonden dan staat er i.p.v. OK de mededeling BAD. We kunnen daarna nogmaals proberen het weg te schrijven stuk opnieuw op te nemen (eventueel op een ander volumeniveau).

Samenvatting hoofdstuk 5

In het eerste deel van dit hoofdstuk zijn programmalussen met de FOR/NEXT constructie behandeld.
Het tweede deel heeft duidelijk gemaakt hoe programma's naar een cassetteband kunnen worden weggeschreven en later weer vanuit de cassetteband kunnen worden geladen.

Multiple choice-vragen bij hoofdstuk 5

A. a. Een programmalus kan alleen worden gerealiseerd met de FOR/NEXT statements.
b. In een FOR/NEXT constructie mag de lopende variabele niet met een negatieve waarde beginnen.

1 uitspraak a is goed en uitspraak b is goed
2 uitspraak a is fout en uitspraak b is goed
3 uitspraak a is goed en uitspraak b is fout
4 uitspraak a is fout en uitspraak b is fout

B. a. Als een FOR/NEXT lus helemaal is doorlopen stopt het programma.
b. Een FOR/NEXT lus wordt altijd volledig doorlopen.

1 uitspraak a is goed en uitspraak b is goed
2 uitspraak a is fout en uitspraak b is goed
3 uitspraak a is goed en uitspraak b is fout
4 uitspraak a is fout en uitspraak b is fout

C. a. Een FOR/NEXT lus mag geen REM statement bevatten.
b. Een FOR/NEXT lus moet in de lus altijd een bepaald uit te voeren statement bevatten.

1 uitspraak a is goed en uitspraak b is goed
2 uitspraak a is fout en uitspraak b is goed
3 uitspraak a is goed en uitspraak b is fout
4 uitspraak a is fout en uitspraak b is fout

D. Welk van de onderstaande FOR statements is fout?

1 FOR P = 57 TO Q STEP − 7.5
2 FOR P = Q TO R STEP S
3 FOR P = Q TO P > R STEP 1
4 FOR P = 2.0 * Q TO 5.0 * R STEP S/2

E. Programma:

```
10 FOR P = 1 TO 10 STEP 2
20 FOR J = P TO 10 STEP 2
30 PRINT P, J
40 NEXT J
50 NEXT P
60 END
```

Als het programma wordt uitgevoerd, doorlopen P en J de volgende waarden:

1 P = 1 en J = 1, 3, 5, . . . dan P = 3 en J = 1, 3, 5, . . .; dan P = 5 en J = 1, 3, 5, . . .; enz.
2 P = 1 en J = 1, 2, 3, . . .; dan P = 2 en J = 1, 2, 3, . . .; dan P = 3 en J = 1, 2, 3, . . .; enz.
3 P = 1, 3, 5, . . . en J = 1; dan P = 1, 3, 5, . . . en J = 3; dan P = 1, 3, 5, . . . en J = 5; enz.
4 P = 1 en J = 1, 3, 5, . . .; dan P = 3 en J = 3, 5, 7, . . .; dan P = 5 en J = 5, 7, 9; enz.

F. We willen de volgende reeks getallen afdrukken:

19 19 19

Welke van de onderstaande programma's voert dit uit?

1
```
10 FOR I = 20 TO 27 STEP 3
20 PRINT 19
30 NEXT I
40 END
```
2
```
10 FOR I = 1 TO 3: LET I = 19
20 PRINT I;
30 NEXT I
40 END
```
3
```
10 FOR I = 0 TO 2
20 PRINT 19,
30 FOR J = 1 TO 22: NEXT J
40 NEXT I
50 END
```
4
```
10 FOR I = 7 TO 12 STEP 2; FOR J = 12 TO 7 STEP -2
20 PRINT I + J,
30 NEXT J: NEXT I
40 END
```

G. Programma:
```
10 FOR X = 1 TO 5
15 PRINT
20 FOR Y = 1 TO 25
25 PRINT
30 FOR Z = 1 TO 10
40 PRINT X, Y, Z, X + Y + Z
50 NEXT Z
60 NEXT Y
70 NEXT X
80 END
```
Hoe vaak wordt het PRINT statement uitgevoerd?

1 1250
2 1252
3 1375
4 1380

H. Programma:
```
10 FOR X = 1 TO 5
20 FOR Y = 1 TO 5
30 FOR Z = 1 TO 5
35 IF Z< = X + Y THEN 45
40 PRINT X, Y, Z, X + Y + Z
45 NEXT Z
50 NEXT Y
55 NEXT X
60 END
```
Hoeveel regels drukt dit programma af?

1 3
2 10
3 19
4 125

I. Programma:

```
 5 LET S = 0
10 FOR I = 1 TO 5
20 FOR J = 1 TO I
30 FOR K = 1 TO J
35 LET S = S + I + J + K
40 NEXT K
50 NEXT J
60 NEXT I
70 PRINT S
80 END
```

Hoe vaak wordt regel 35 uitgevoerd?

1 125 keer
2 35 keer
3 7 keer
4 1 keer

J. Welke waarde wordt er in het programma van vraag I afgedrukt?

1 0
2 286
3 300
4 315

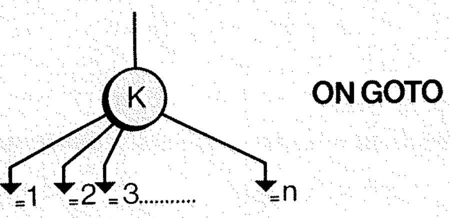
Hoofdstuk 6

Subroutines

In dit hoofdstuk wordt het gebruik van statements, die meervoudige vertakkingen op een bepaald punt in een programma mogelijk maken, besproken. Daarnaast wordt er veel aandacht besteed aan het gebruik van subroutines. Door deze in programma's toe te passen, kan een mooie (dus ook eenvoudig te realiseren en te testen) programmastructuur worden verkregen.

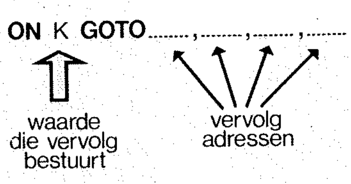
6.1. Meervoudige sprongen

Door middel van het IF. THEN statement zijn we in staat om in een BASIC programma, afhankelijk van de waarde van een conditie, naar een bepaald programmadeel te springen. In veel gevallen komt het echter voor dat we aan een splitsing in twee richtingen (zoals bij de IF. THEN) niet voldoende hebben.
We willen in meerdere richtingen kunnen springen. Deze meervoudige sprongen hebben de naam *multiple branch* of *multiway branch* gekregen. In afbeelding 6.1 vinden we een schematische weergave van de meervoudige sprong. Hierbij wordt door de waarde K aangegeven waar we naar toe willen. Voor K = 1 gaan we in de richting waarbij = 1 staat, voor K = 2 gaan we in de richting = 2, enz.



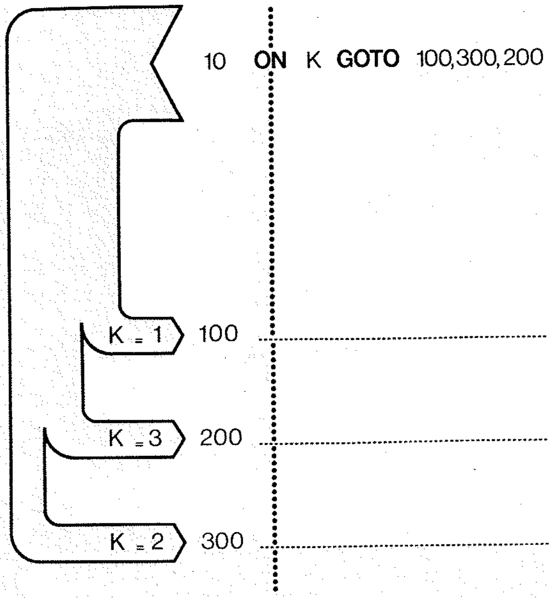
Afb. 6.1. Flow-diagram van een ON. GOTO statement

In BASIC hebben we voor deze constructie het ON. GOTO statement. Achter het keyword ON staat de variabele die de sprong zal besturen en achter het keyword GOTO staat een lijst met regelnummers waar we respectievelijk voor K = 1, 2, 3, 4, enz. naar toe springen. In afbeeldingen 6.2 en 6.3 vinden we het ON. GOTO schematisch terug in een voorbeeldprogramma.
Het ON. GOTO statement wordt in de praktijk ook 'computed GOTO' genoemd, omdat de waarde van de variabele tijdens het programma kan worden veranderd. We kiezen dan met de berekende waarde een vervolgrichting in het programma.
In het voorbeeldprogramma van afbeelding 6.4 vinden we een toepassing van het ON. GOTO statement. We nemen weer het programma om *koersen* om te zetten. Aan het einde van het programma vraagt de computer ons wat we verder willen doen. De regels 100/150 vertellen wat de mogelijkheden zijn. Dan moeten we in regel 160 een getal invoeren dat aangeeft wat onze keuze is. Als we de waarde 1 intikken betekent dit dat we willen stoppen. De computer zal daarbij door moeten gaan met het END statement op regel 180. Bij het intikken van een 2 vragen we om een extra kopie van de zojuist gemaakte tabel. De computer zal bij regel 50 weer moeten beginnen. Dit is namelijk het punt waar de invoerdata waren in-



Afb. 6.2. Opbouw van het ON. GOTO statement

gevoerd. Deze data zijn nog beschikbaar in de variabelen B, E en S, zodat we er opnieuw gebruik van kunnen maken.
Indien we echter nieuwe data willen invoeren, moet het programma op regel 30 worden herhaald. Het ON. GOTO statement, dat we hierbij toepassen, krijgt als variabele die, welke ook wordt gebruikt bij het INPUT statement en als sprongadressen 180, 50 en 30.



Afb. 6.3. Sprongen in een programma met het ON. GOTO statement

```
*LIST
10     PRINT "KØERSØMZETTING"
15     PRINT "==*=*==*==*=*=="
20     PRINT
30     INPUT "$1 IS IN GULDENS";A
31     PRINT
32     INPUT "BEGIN TABEL = $";B
33     PRINT
34     INPUT "EINDE TABEL = $";E
35     PRINT
36     INPUT "STAPGRØØTTE = $";S
37     PRINT
40     PRINT
50     LET I=B
60     IF I>E THEN 100
70     PRINT "$";I;" = fl";INT(I*A*100.0+0.5)/100.0
80     LET I=I+S
90     GØTØ 60
100    PRINT
110    PRINT "VØER IN 1,2 ØF 3"
120    PRINT
130    PRINT "1: STØP PRØGRAMMA"
140    PRINT "2: EXTRA CØPY VAN DE TABEL"
150    PRINT "3: NIEUWE GEGEVENS"
```

Afb. 6.4. Programma met toepassing van het ON. GOTO statement

(vervolg van het programma)

```
160    INPUT K
170    ØN K GØTØ 180,50,30
180    END
*RUN

KØERSØMZETTING
==**====**===**==

$1 IS IN GULDENS?1.99
BEGIN TABEL = $?1
EINDE TABEL = $?10
STAPGRØØTTE = $?1

$ 1.0 = fl 1.99
$ 2.0 = fl 3.98
$ 3.0 = fl 5.97
$ 4.0 = fl 7.96
$ 5.0 = fl 9.95
$ 6.0 = fl 11.94
$ 7.0 = fl 13.93
$ 8.0 = fl 15.92
$ 9.0 = fl 17.91
$ 10.0 = fl 19.9

VØER IN 1,2 ØF 3

1: STØP PRØGRAMMA
2: EXTRA CØPY VAN DE TABEL
3: NIEUWE GEGEVENS
?2$ 1.0 = fl 1.99
$ 2.0 = fl 3.98
$ 3.0 = fl 5.97
$ 4.0 = fl 7.96
$ 5.0 = fl 9.95
$ 6.0 = fl 11.94
$ 7.0 = fl 13.93
$ 8.0 = fl 15.92
$ 9.0 = fl 17.91
$ 10.0 = fl 19.9

VØER IN 1,2 ØF 3
```

(vervolg van het programma)

```
1: STØP PRØGRAMMA
2: EXTRA CØPY VAN DE TABEL
3: NIEUWE GEGEVENS
?3$1 IS IN GULDENS?1.96
BEGIN TABEL = $?5
EINDE TABEL = $?25
STAPGRØØTTE = $?5

$ 5.0 = fl 9.8
$ 10.0 = fl 19.6
$ 15.0 = fl 29.4
$ 20.0 = fl 39.2
$ 25.0 = fl 49.0

VØER IN 1,2 ØF 3

1: STØP PRØGRAMMA
2: EXTRA CØPY VAN DE TABEL
3: NIEUWE GEGEVENS
?1END PRØGRAM
*
```

Statement	Effect
ON. GOTO	Meervoudig sprongstatement.

Algemene vorm:
regelnummer ON x GOTO $s_1, s_2, \ldots, s_n$

waarbij x een numerieke constante, een numerieke variabele of een rekenkundige expressie is en $s_1, s_2, \ldots, s_n$ regelnummers zijn.

Effect:
In het meervoudig sprongstatement wordt een sprong uitgevoerd naar een regelnummer. Het regelnummer, waarheen wordt gesprongen, wordt bepaald door de integer waarde van x. Voor x = 1, 2, 3, . . . wordt gesprongen naar respectievelijk regelnummer $s_1, s_2, \ldots, s_n$.
Als het integer deel van x kleiner is dan 1 of groter dan het aantal sprongadressen, wordt verder gegaan met de regel, die volgt op het ON. GOTO statement.

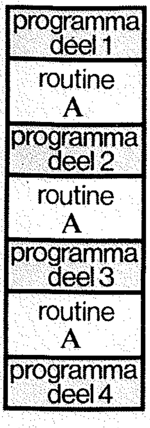
NB. 1
Met het integer deel van een getalwaarde wordt bedoeld dat de waarde wordt geschreven zonder exponent en dan wordt afgebroken achter de decimale punt.

NB. 2
Bij sommige computers mag x niet buiten de lijst van sprongadressen vallen. Gebeurt dit wel dan zal er een foutmelding worden gegeven.

6.2. Subroutines (GOSUB, RETURN, ON. GOSUB)

Wanneer een bepaalde bewerking (een berekening of een I/O-programma) in een programma meerdere keren voorkomt en we geen gebruik kunnen maken van programmalussen, is het erg tijdrovend om steeds weer dezelfde statements te moeten opschrijven. Het gebruik van *subroutines* voorziet in dit probleem.
Een subroutine is een deel van een programma, dat we slechts een keer hoeven te schrijven, maar daarna op verschillende plaatsen kunnen aanroepen. In de subroutine zal een bepaalde bewerking (meerdere functioneel bij elkaar behorende operaties) worden gerealiseerd. Dit kunnen complexe wiskundige functies, I/O-programma's e.d. zijn.
Een subroutine vervangt als het ware een hoofdbewerkingsblok uit een flow-diagram. We kunnen een programma opbouwen uit een reeks achter elkaar aangeroepen subroutines. Deze subroutines worden in BASIC aan het eind van het programma toegevoegd. Bij de executie wordt dan vanuit het hoofdprogramma naar de opgegeven subroutine gesprongen (in BASIC met het GOSUB statement). De subroutine wordt achtereenvolgens uitgevoerd en daarna wordt vanuit de subroutine (door middel van het RETURN statement)

teruggesprongen naar de regel, die volgt op de regel waarin de subroutine werd aangeroepen.
Afbeelding 6.5 laat een programma zien waarin driemaal de routine A wordt gebruikt. In afbeelding 6.6 wordt de routine A als subroutine uit het programma gehaald en eenmaal achteraan toegevoegd. Op de plaatsen waar de routine stond, wordt via een GOSUB statement toch naar de routine gesprongen en vanuit de RETURN in de subroutine keren we terug naar de plaats direct achter het aanroepende GOSUB statement. Afbeelding 6.7 geeft een symbool voor een subroutine.
We behandelen nu eerst de twee nieuwe statements die speciaal voor het toepassen van subroutines aan BASIC zijn toegevoegd.



Afb. 6.5. Programma dat driemaal de routine A uitvoert

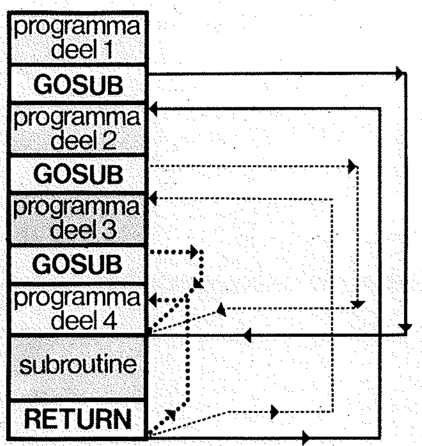
6.3. GOSUB en RETURN statement

Een subroutine wordt aan het einde van een programma geplaatst en wel voor het DATA statement (uitleg hiervan volgt in hoofdstuk 7) en END statement. Doorgaans voeren we een programma uit volgens de regels die in de voorgaande hoofdstukken zijn gegeven. Zodra echter op een regel het statement GOSUB wordt gevonden, zal er een aanroep plaatsvinden van de subroutine, die wordt gespecificeerd door het regelnummer dat direct op het keyword GOSUB volgt.
De besturing van de computer onthoudt vanuit welke regel de aanroep van de subroutine plaatsvond. Daarna gaat de executie door op de regel waarmee de subroutine begint.

Voorbeeld

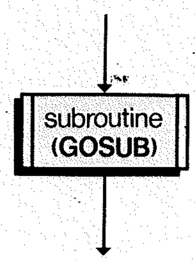
```
100 GOSUB 1000
```

betekent dat er een subroutine wordt aangeroepen, die begint op regel 1000.



Afb. 6.6. Programma met routine A als subroutine

Het aanroepen van een subroutine houdt in dat tijdens de op de plaats van de oproep een programmadeel wordt tussengevoegd. Na de aanroep van de subroutine wordt verder gegaan met het uitvoeren in normale volgorde van de subroutine-statements. We moeten er echter wel op letten dat we (via al of niet conditionele sprongen) niet uit de subroutine springen. De subroutine mag namelijk slechts verlaten worden via een RETURN statement. Deze uitgang is het statement RETURN.
De uitvoering hiervan heeft tot gevolg dat wordt teruggegaan naar het statement, dat volgt op de GOSUB die als laatste was uitgevoerd.
Aan variabelen, die binnen een subroutine worden gebruikt, kunnen in het stuk programma dat direct voor de aanroep (GOSUB) staat, parameters worden overgedragen. Het resultaat van de bewerking in de subroutine kan op dezelfde manier worden overgedragen aan het aanroepende programma. Dit heet het overdragen van argumenten en resultaten. We moeten hierbij wel dezelfde namen voor de variabelen gebruiken in het aanroepende programma en de subroutine.

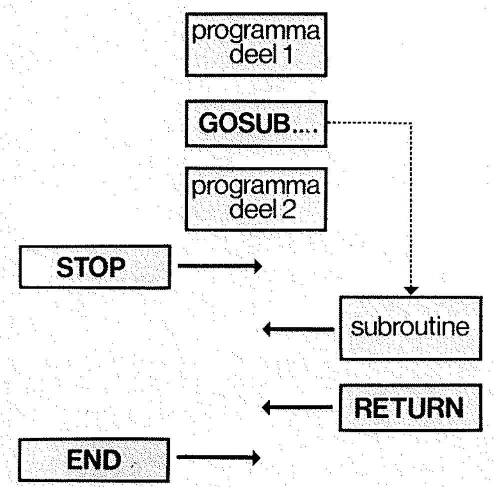


Afb. 6.7. Symbool van een subroutine

N.B. Er mogen weliswaar meerdere RETURN statements in een subroutine voorkomen, maar dit is af te raden omdat dit tot onoverzichtelijke programmastructuur kan leiden.

6.4. STOP statement

Het is noodzakelijk om het hoofdprogramma te *scheiden* van de subroutines. De computer gaat anders na het hoofdprogramma door met de subroutines en de gevolgen zouden niet te definiëren zijn zodra we een RETURN statement tegenkomen omdat de computer niet zou weten waarheen hij moet terugkeren.
Om de executie van het programma correct te stoppen, kunnen we achter het hoofdprogramma en voor de subroutines een STOP statement zetten. Het programma zal dan op deze plaats stoppen. Het END statement, dat normaal voor het stoppen zorgt, heeft nu als functie het aangeven van het einde van de reeks opgeschreven statements.



Afb. 6.8. Plaats van de statements GOSUB, RETURN, END en STOP

In afbeelding 6.8 wordt de plaatsing van de statements nog eens duidelijk gemaakt. Vaak wordt het regelnummer vermeld waar werd gestopt. Dit is makkelijk bij het lokaliseren van fouten door middel van het tussenvoegen van STOP statements.
We kunnen in plaats van een STOP statement ook een GOTO bij het regelnummer waar het END statement staat tussenvoegen. Het effect is daarbij gelijk.
In afbeelding 6.9 staat een programmavoorbeeld waarin de subroutines worden toegepast voor het maken van een *histogram* (grafiek bestaande uit balken) voor de functie $y = x^2 + 2$.

```
*LIST
100   REM HISTØGRAM VØØR Y=X↑2+2
110   REM
120   REM ----------------------
130   PRINT
135   PRINT "               Y --->"
140   GØSUB 600
145   FØR X=-5.0 TØ 5.0
150   PRINT "X = ";X;":";
160   LET Y=X*X+2.0
170   GØSUB 500
```

Afb. 6.9. Programma met subroutines voor balken en assen van een histogram

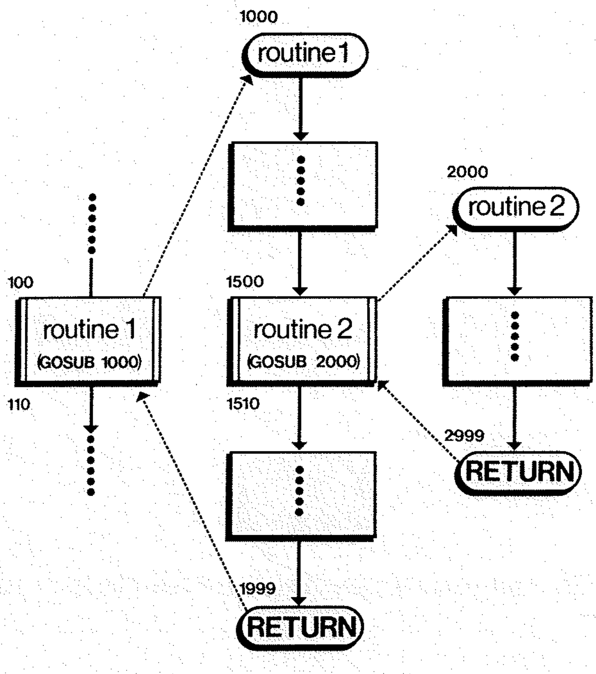
(vervolg programma)
```
180   NEXT X
185   GØSUB 600
190   STØP
500   REM PRINT DE WAARDE VAN Y
505   REM
506   REM ----------------------
510   FØR I=1.0 TØ Y
520   PRINT "*";
530   NEXT I
540   PRINT
550   RETURN
600   REM PRINT DE AS
601   REM
605   REM ----------------------
610   PRINT "         :----:----:----:----:----:----:----:----
620   RETURN
900   END
*RUN

              Y --->
          :----:----:----:----:----:----:----:----:
X = -5.0:***************************
X = -4.0:******************
X = -3.0:***********
X = -2.0:******
X = -1.0:***
X =  0.0:**
X =  1.0:***
X =  2.0:******
X =  3.0:***********
X =  4.0:******************
X =  5.0:***************************
          :----:----:----:----:----:----:----:----:
STØPPED IN LINE 190
*
```

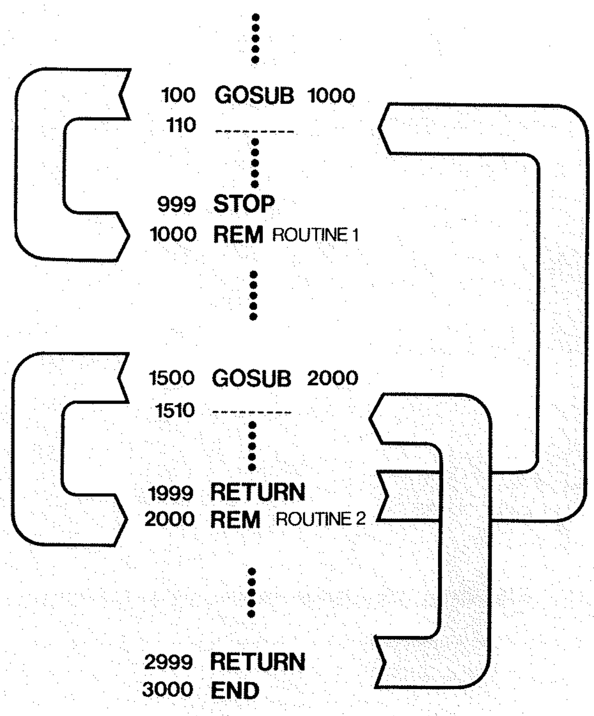
6.5. Nesten van subroutines

Evenals bij de FOR/NEXT combinatie kunnen ook subroutines worden genest. Dit betekent dat in een subroutine weer een tweede subroutine kan worden aangeroepen. In de afbeeldingen 6.10 en 6.11 zien we achtereenvolgens een flow-diagram en het bijbehorende programma, waar in een subroutine weer een subroutine wordt aangeroepen.
Op regel 100 wordt door de GOSUB de subroutine op regel 1000 aangeroepen. Dan volgt de executie van de subroutine 1 tot aan regel 1500. De GOSUB, die hier staat, roept de subroutine 2 aan op regel 2000. Na de uitvoering van deze laatste routines komen we via het RETURN statement (2999) terug op regel 1510 van subroutine 1. Deze wordt afgewerkt tot aan regel 1999. Deze regel stuurt de computerbesturing via de RETURN terug naar regel 110 van het hoofdprogramma.

Het nut van het nesten van subroutines en het toepassen van subroutines in het algemeen is het voordeel dat we ondervinden, doordat we een flow-diagram, dat uit grove blokken is opgebouwd, vrij eenvoudig in een programma kunnen omzetten. De grove blokken zijn namelijk vaak als functioneel blok te zien en kunnen daarom als subroutine worden geprogrammeerd. Hierdoor krijgen we ook in het programma een opbouw in blokken, wat de leesbaarheid, de testbaarheid en de betrouwbaarheid meestal ten goede zal komen.
Het aantal niveaus waarin genest kan worden, is afhankelijk van het type computer. Het is echter altijd beperkt.



Afb. 6.10 Nesten van subroutines

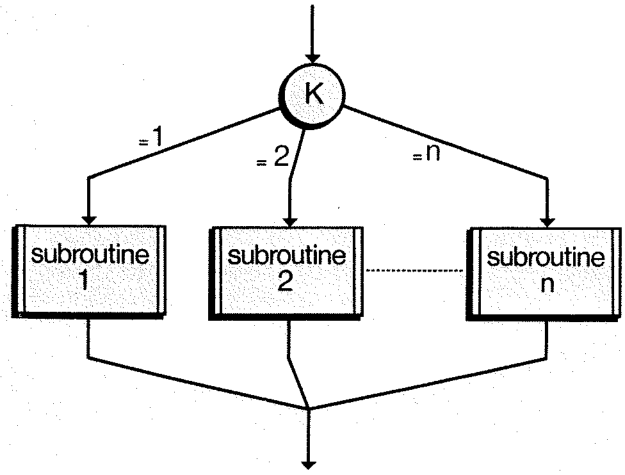


Afb. 6.11. Volgorde van afwikkeling bij nesten van subroutines

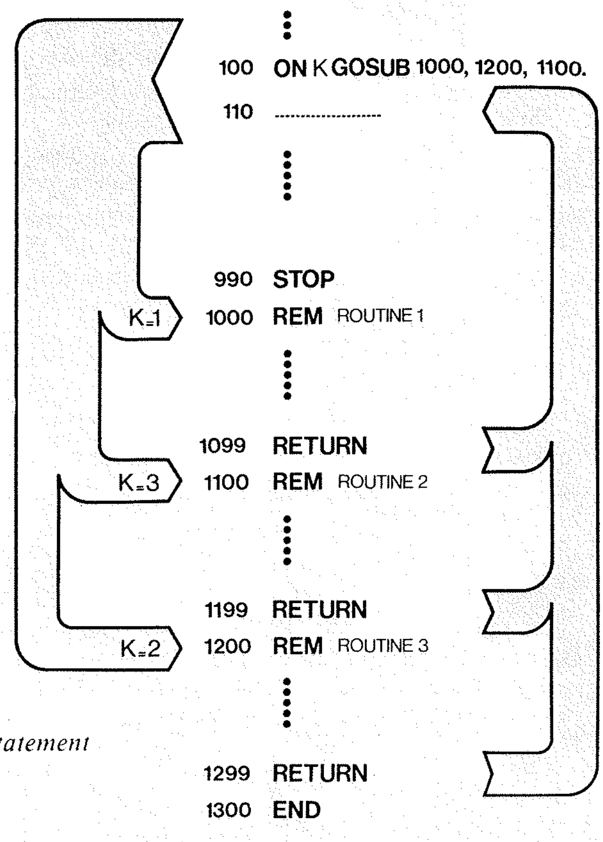
6.6. ON. GOSUB statement

Een uitbreiding van het GOSUB statement is het ON. GOSUB statement. Bij de GOSUB/RETURN combinatie werd een bepaalde subroutine aangeroepen, nl. niet conditioneel. Als we met de GOSUB dezelfde constructie maken als met de ON. GOTO krijgen we een conditionele subroutine-aanroep. Hierbij wordt in het programma op een bepaalde plaats de mogelijkheid geboden om te *kiezen* uit enige subroutines. De waarde van een variabele of rekenkundige expressie zal bepalen welke subroutine wordt aangeroepen. Als de waarde 1 is, wordt de subroutine aangeroepen, die hoort bij het eerste regelnummer dat op het keyword GOSUB volgt; voor de waarde 2 gaan we naar de subroutine die op het tweede regelnummer van de lijst begint; enz.

Het verloop van de keuze is identiek aan het kiezen bij het ON. GOTO statement. Hier hebben we echter een subroutine-aanroep



Afb. 6.12. Een flow-diagram van de ON. GOSUB constructie



Afb. 6.13. Afwikkelingsvolgorde bij het ON. GOSUB statement

met later een RETURN direct achter de aanroepende ON. GOSUB. Afbeeldingen 6.12 en 6.13 laten weer het flow-diagram en een voorbeeld in de vorm van een programma zien.
In afbeelding 6.14 zijn een aantal constructies toegepast om te demonstreren hoe de FOR/NEXT, IF. THEN en ON. GOSUB statements in de praktijk worden toegepast.
In het.programma worden uit een viertal in te voeren letters alle combinaties van letters gemaakt. Het is een vereenvoudigde vorm van het spel Scrabble.
In de regels 10 tot en met 53 voeren we na enige verklarende tekst de vier letters in. Deze worden in het INPUT statement toegekend aan de variabelen A$, B$, C$ en D$.
Eerst maken we woorden van één letter (het zijn nauwelijks woorden te noemen, maar voor de volledigheid noemen we ze toch). Hiervoor laten we de variabele I lopen van 1 tot en met 4 in een FOR/NEXT lus. Binnen deze lus roepen we een subroutine aan. Welke subroutine we aanroepen wordt bepaald door de waarde van de variabele I. Voor I = 1 roepen we de routine 800 op en hierin wordt de letter afgedrukt, die hoort bij de variabele A$. Voor I = 2 drukken we de waarde van de variabele B$ af, enz.
Nadat alle combinaties van één letter zijn afgedrukt, komen in regels 190 t/m 270 de woorden van twee letters aan de orde. We gebruiken hiervoor twee FOR/NEXT lussen, die elkaar helemaal omvatten. Namelijk voor I van 1 t/m 4 en voor J van 1 t/m 4. In regel 220 testen we of we tweemaal dezelfde letter kiezen, wat natuurlijk nooit mag.

Afb. 6.14. Programma dat woorden maakt van combinaties van 4 letters

```
*LIST
10    REM -SCRABBLE PRØGRAMMA VØØR 4 LETTERS
11    REM
30    PRINT "VØER VIER LETTERS IN"
31    PRINT "WELKE DØØR KØMMA'S GESCHEIDEN ZIJN"
33    INPUT A$,B$,C$,D$
40    PRINT
50    PRINT "ALLE CØMBINATIES VAN DE VIER LETTERS:"
51    PRINT A$,B$,C$,D$
53    PRINT "ZIJN:"
54    PRINT
100   PRINT "WØØRDEN VAN EEN LETTER"
105   PRINT
110   FØR I=I TØ 4.0
120   ØN I GØSUB 800,810,820,830
130   PRINT
140   NEXT I
150   PRINT
190   PRINT "WØØRDEN VAN TWEE LETTERS"
195   PRINT
200   FØR I=1.0 TØ 4.0
210   FØR J=1.0 TØ 4.0
220   IF I=J GØTØ 250
230   ØN I GØSUB 800,810,820,830
240   ØN J GØSUB 800,810,820,830
245   PRINT
250   NEXT J
260   NEXT I
270   PRINT
280   PRINT "WØØRDEN VAN DRIE LETTERS"
290   PRINT
300   FØR I=1.0 TØ 4.0
310   FØR J=1.0 TØ 4.0
315   IF I=J GØTØ 390
320   FØR K=1.0 TØ 4.0
330   IF I=K ØR J=K GØTØ 380
340   ØN I GØSUB 800,810,820,830
350   ØN J GØSUB 800,810,820,830
360   ØN K GØSUB 800,810,820,830
370   PRINT
380   NEXT K
390   NEXT J
400   NEXT I
410   PRINT
420   PRINT "WØØRDEN VAN VIER LETTERS"
430   PRINT
500   FØR I=1.0 TØ 4.0
510   FØR J=1.0 TØ 4.0
515   IF I=J GØTØ 630
520   FØR K=1.0 TØ 4.0
525   IF I=K ØR J=K GØTØ 620
530   FØR L=1.0 TØ 4.0
540   IF I=L ØR J=L ØR K=L GØTØ 610
```

(vervolg programma)

```
550   ØN I GØSUB 800,810,820,830
560   ØN J GØSUB 800,810,820,830
570   ØN K GØSUB 800,810,820,830
580   ØN L GØSUB 800,810,820,830
590   PRINT
610   NEXT L
620   NEXT K
630   NEXT J
640   NEXT I
799   STØP
800   PRINT A$;
805   RETURN
810   PRINT B$;
815   RETURN
820   PRINT C$;
825   RETURN
830   PRINT D$;
835   RETURN
900   END
```

```
*RUN
VØER VIER LETTERS IN
WELKE DØØR KØMMA'S GESCHEIDEN ZIJN
?P,Ø,S,T
ALLE CØMBINATIES VAN DE VIER LETTERS:
P           Ø           S           T
ZIJN:

WØØRDEN VAN EEN LETTER

P
Ø
S
T

WØØRDEN VAN TWEE LETTERS

PØ
PS
PT
ØP
ØS
ØT
SP
SØ
ST
TP
TØ
TS

WØØRDEN VAN DRIE LETTERS

PØS
PØT
PSØ
PST
PTØ
PTS
ØPS
ØPT
ØSP
ØST
ØTP
ØTS
SPØ
SPT
SØP
SØT
```

(vervolg output)

```
STP
STØ
TPØ
TPS
TØP
TØS
TSP
TSØ

WØØRDEN VAN VIER LETTERS

PØST
PØTS
PSØT
PSTØ
PTØS
PTSØ
ØPST
ØPTS
ØSPT
ØSTP
ØTPS
ØTSP
SPØT
SPTØ
SØPT
SØTP
STPØ
STØP
TPØS
TPSØ
TØPS
TØSP
TSPØ
TSØP
STØPPED IN LINE 799
*
```

Afb. 6.15. Output van het programma van afb. 6.14.

Het vervolg van het programma verloopt op soortgelijke wijze. Het verschil is dat we steeds meer lussen om elkaar heen krijgen en steeds meer combinaties op gelijkheid moeten onderzoeken.

Een mooie oefening zou zijn om eens een keer met zeven letters te werken zoals in werkelijkheid bij het spel Scrabble gebeurt. Het programma kan heel lang duren. Er zijn namelijk 13 699 combinaties mogelijk.
In afbeelding 6.15 vinden we de resultaten voor de vier letters, te beginnen met het woord

POST

Statement	Effect
GOSUB	Niet-conditionele subroutine-aanroep

Algemene vorm:

regelnummer GOSUB n

waarbij n een regelnummer van het BASIC programma is.

Effect:

De niet-conditionele aanroep van een subroutine gebeurt met het BASIC statement GOSUB. Hiermee wordt aan BASIC een handig gereedschap toegevoegd, waardoor reeksen handelingen, die op meerdere plaatsen in het programma voorkomen, kunnen worden geprogrammeerd zonder ze elke keer opnieuw te moeten schrijven. Bij de executie van het GOSUB statement wordt de subroutine aangeroepen, die op het regelnummer begint dat volgt op het keyword GOSUB.
Het programmablok dat in de subroutine is opgenomen, moet worden afgesloten met een RETURN statement. Bij de executie van het RETURN statement wordt teruggekeerd naar het statement, dat volgt op het aanroepende GOSUB statement.

Statement	Effect
RETURN	Terugkeer van een subroutine

Algemene vorm:

regelnummer RETURN

Effect:

Door middel van een RETURN statement moet een subroutine worden verlaten. Er wordt dan teruggekeerd naar het aanroepende programma op die regel, die direct volgt op het GOSUB of ON. GOSUB statement, waar de zojuist beëindigde subroutine is aangeroepen.

Statement	Effect
STOP	Beëindiging van de programma-executie. (Scheiding tussen hoofdprogramma en data of subroutines, of gebruik voor testdoeleinden.)

Algemene vorm:
regelnummer STOP

Effect:
Door de executie van het STOP statement wordt de executie van het BASIC programma afgebroken.

Statement	Effect
ON. GOSUB	Conditionele subroutine-aanroep.

Algemene vorm:

regelnummer ON x GOSUB $s_1, s_2, \ldots, s_n$
waarbij x een numerieke constante, een numerieke variabele of een rekenkundige expressie is en $s_1, s_2, \ldots, s_n$ regelnummers zijn van het BASIC programma.

Effect:
Het conditionele GOSUB statement zorgt ervoor dat in een programma op een bepaald punt één van een reeks subroutines kan worden gekozen. In de lijst met regelnummers, die achter het keyword GOSUB staat, worden de regelnummers gegeven van de beginpunten van de subroutines. Afhankelijk van de waarde van x (afgebroken naar een integer waarde, zie ook ON, GOTO) zal voor x = 1, 2, 3, ... worden verder gegaan met resp. $s_1, s_2, s_3 \ldots$ Indien de integer waarde kleiner dan 1 is, wordt doorgegaan met het statement, dat direct volgt op het ON. GOSUB statement. Zodra in de aangeroepen subroutine een RETURN statement voorkomt, wordt naar het statement teruggekeerd dat direct volgt op het aanroepende ON. GOSUB statement.
Voor x gelden dezelfde voorwaarden als bij het ON. GOTO statement.

Samenvatting hoofdstuk 6

In dit hoofdstuk zijn een aantal gereedschappen geleverd om een structuur in de programma's te kunnen aanbrengen. Dit blijkt in de praktijk een pad naar een zindelijke manier van programmeren te zijn. Voor de gebruiker van de computer heeft dit tot gevolg dat er minder fouten zullen worden gemaakt en dat er minder tijd nodig zal blijken te zijn voor het opzetten van de programma's.

Multiple choice-vragen bij hoofdstuk 6

A. **a.** Door het toepassen van subroutines worden programma's beter wat betreft hun structuur.
b. Het toepassen van subroutines levert wat betreft het aantal regels meestal kortere programma's.

1 uitspraak a is goed en uitspraak b is goed
2 uitspraak a is fout en uitspraak b is goed
3 uitspraak a is goed en uitspraak b is fout
4 uitspraak a is fout en uitspraak b is fout

B. De functie van het ON. GOTO statement is:

1 een programmalus herhaald te doorlopen
2 afhankelijk van de numerieke waarde tussen ON en GOTO een vervolgrichting voor het programma te bepalen
3 te testen of er een sprong wordt gemaakt en dan het regelnummer af te drukken waar naar toe wordt gesprongen
4 te testen of de bijbehorende FOR/NEXT lus moet worden beëindigd

C. Wat is er fout in het onderstaand statement?

ON K > = 3 GOTO 100, 100, 200, 200, 100

1 er mag nooit een relationele expressie staan tussen ON en GOTO
2 de reeks met regelnummers mag nooit meer keren hetzelfde regelnummer bevatten
3 K kan een gebroken getal zijn
4 er mogen maar drie regelnummers achter GOTO staan

D. Programma:

```
 10  INPUT A
 20  IF A=0 THEN K=2
 40  IF A>0 THEN K=3
 40  IF A<0 THEN K=1
 50  ON K GOTO 100, 200 300
 60  PRINT "SIGN=-1"
110  GOTO 400
200  PRINT "SIGN=0"
210  GOTO 400
300  PRINT "SIGN=1"
400  END
```

Welke bewering is juist?

1 als A<0 worden zowel de regels 100, 200, als 300 uitgevoerd
2 het programma kan niet werken omdat in de regels 20, 30 en 40 alle mogelijkheden voor A voorkomen
3 het programma geeft als resultaat of A negatief, nul of positief is, door respectievelijk de tekst SIGN = −1, SIGN = 0 of SIGN = 1 af te drukken
4 het programma is fout, omdat nooit naar regel 10 wordt teruggesprongen

E. **a.** Tijdens de executiefase van een programma mag het RETURN statement nooit eerder worden uitgevoerd dan een subroutine-aanroep.

b. Als in een subroutine meer RETURN statements voorkomen, krijgen we een foutmelding.

1 uitspraak a is goed en uitspraak b is goed
2 uitspraak a is fout en uitspraak b is goed
3 uitspraak a is goed en uitspraak b is fout
4 uitspraak a is fout en uitspraak b is fout

F. Wat is het verschil tussen de GOSUB en de GOTO statements?

1 het GOSUB statement wordt gebruikt voor het maken van programmalussen en het GOTO statement niet
2 het GOSUB statement is sneller dan het GOTO statement
3 het GOSUB statement verzorgt de aanroep van een subroutine en het GOTO statement niet
4 via de GOTO springen we naar een subroutine en keren er niet uit terug en bij een aanroep via het GOSUB statement keren we wel uit de subroutine terug

G. Programma:

```
 10  LET A = 1
 20  GOSUB 100
 30  GOSUB 100
 40  GOSUB 100
 50  PRINT A
 60  STOP
100  LET A = A*10
110  RETURN
999  END
```

Welke waarde wordt voor A afgedrukt?

1 1
2 10
3 30
4 1000

H. Programma

```
 10  LET A = 50: LET B = 100
 20  GOSUB 100
 30  GOSUB 200
 40  PRINT A, B
 50  STOP
100  LET A = A + B
110  GOSUB 200
120  RETURN
200  LET A = A/5
210  RETURN
999  END
```

Welke waarden worden er afgedrukt?

1 6 100
2 30 100
3 75 100
4 150 100

I. **a.** Door het gebruik van het ON-GOSUB statement wordt het mogelijk om op een plaats in een programma een keuze uit een reeks van subroutines te maken.
b. De waarde die tussen ON en GOSUB staat moet een geheel getal zijn.

1 uitspraak a is goed en uitspraak b is goed
2 uitspraak a is fout en uitspraak b is goed
3 uitspraak a is goed en uitspraak b is fout
4 uitspraak a is fout en uitspraak b is fout

J. Programma:

```
 10   INPUT M
 20   IF M > =0 OR M> =5 THEN GOTO 10
 30   ON M GOSUB 100, 200, 300, 400
 40   PRINT N$
 60   STOP
100   LET N$='EEN'
110   RETURN
200   LET N$='TWEE'
210   RETURN
400   LET N$='DRIE'
310   RETURN
400   LET N$='VIER'
410   RETURN
999   END
```

Er wordt achtereenvolgens ingevoerd:

```
RUN
-4
 4.2
```

Wat wordt er afgedrukt?

1 helemaal niets
2 VIER
VIER
3 VIER
4 EEN
VIER

Hoofdstuk 7

Datastructuren in BASIC

In dit hoofdstuk zien we hoe we gegevens kunnen invoeren zonder dat die gegevens steeds opnieuw worden ingetypt. Daarom zullen de BASIC statements DATA, READ en RESTORE worden behandeld. Daarna gaan we in op het werken met tabellen via de begrippen array en het daarbij horende BASIC statement DIM. We zullen ook zien hoe we in een array elementen kunnen selecteren. Tevens zien we hoe we variabelen op cassette kunnen wegschrijven en teruglezen met SAVEA en LOADA.

7.1. Inleiding

Wanneer we meer gegevens in de computer willen invoeren, kunnen we het INPUT statement gebruiken. Bij de *executie* van het programma kost dit tijd, want we moeten de waarden steeds intypen. De computer wacht dan tot iemand iets intypt. Voor grotere hoeveelheden gegevens, die steeds dezelfde waarden aannemen, is dat onaangenaam werk. Bovendien is de kans op typefouten groot, zodat daardoor ook verkeerde resultaten gaan optreden. Het INPUT statement, dat we hierbij gebruiken is uitgebreid behandeld in hoofdstuk 3. We weten dus dat INPUT A ervoor zorgt dat de waarde voor A door de gebruiker van het programma moet worden ingetypt.
Het is bij de Teleac/DAI personal computer ook mogelijk de ingetypte waarden op een cassetterecorderband op te slaan en later onder programmabesturing weer op te halen. In dit hoofdstuk en in hoofdstuk 19 zullen we daarop ingaan.
Wanneer het om weinig gegevens met steeds dezelfde waarden gaat, kunnen deze waarden als variabelen in het programma worden geïntroduceerd door middel van het LET statement. In hoofdstuk 3 zagen we overigens dat LET statements ook weggelaten mogen worden, mits duidelijk is dat er een toekenning van een waarde wordt bedoeld.

Voorbeeld

```
*LIST
100   LET A=15.0
110   N=12.0
120   M=132.0
130   PRINT A,N,M
*RUN
 15.0          12.0          132.0
*
```

Opmerking: Programma's moeten normaal worden afgesloten met een END statement, maar bij sommige computers is dit niet noodzakelijk.

Nadat het programmagedeelte is uitgevoerd, hebben de variabelen A, N en M respectievelijk de waarden 15, 12 en 132 gekregen. We kunnen dan vervolgens in het programma daarmee rekening houden.

```
*LIST
100   LET A=15.0
110   N=12.0
120   M=132.0
130   PRINT A,N,M
140   Z=A+M+N
150   PRINT Z
*RUN
 15.0         12.0          132.0
 159.0
*
```

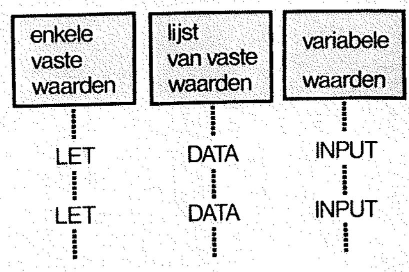
Dus Z heeft na uitvoering de waarde 159 gekregen. Daarna kunnen we opnieuw waarden gaan toekennen en rekenen.

```
*LIST
100   LET A=15.0
110   N=12.0
120   M=132.0
130   PRINT A,N,M
140   Z=A+M+N
150   PRINT Z
160   A=16.0
170   M=2.0
180   Z=A+N+M
190   PRINT Z
*RUN
 15.0         12.0          132.0
 159.0
 30.0
*
```

Na uitvoering van deze statements heeft Z de waarde 30 gekregen. Want N had nog de waarde 12. Deze methode is niet erg aantrekkelijk wanneer het om veel variabelen gaat, die moeten worden toegekend.

7.2. Het statement DATA

Het is dan ook veel aantrekkelijker deze waarden van tevoren een keer op te bergen en naar behoefte op te halen. We maken als het ware een lijst van de voorkomende waarden. Iedere keer als we een waarde nodig hebben lezen we die waarde uit de lijst. In BASIC kunnen we een dergelijke lijst samenstellen met behulp van het statement DATA (zie afbeelding 7.1).



Afb. 7.1. Invoermogelijkheden in een BASIC programma

```
*LIST
10    DATA 1,3,0.3,0.4
*
```

Na executie van dit statement is er in de computer een lijst opgebouwd, die achtereenvolgens bestaat uit de waarden:

1 3 0.3 0.4

Deze gegevens kunnen later met behulp van het READ statement uit de lijst worden gelezen.

Statement	Effect
DATA	Gegevens worden in een datagebied opgeborgen.

Algemene vorm:

regelnummer DATA x_1 [, x_2, x_3, . . ., x_n]

waarbij x_1 tot en met x_n numerieke constanten of strings zijn.

Dus zoals gewoonlijk eerst het regelnummer, dan een of meer blanco's, vervolgens het woord DATA en daarna een of meerdere blanco's, gevolgd door tenminste één constante.

Effect:

De gegevens, die op DATA volgen, worden in één datagebied opgeborgen. Met behulp van een READ statement kunnen deze waarden één voor één uit het datagebied worden gehaald.
Wanneer meerdere *constanten* worden gebruikt, moeten deze door *komma's* gescheiden zijn. Iedere constante mag een geheel getal zijn (bijvoorbeeld 3) of een gebroken getal (bijvoorbeeld 0.3). Het laatste getal mag niet worden gevolgd door een komma! Dit zou namelijk veronderstellen dat er nog een getal volgt. In plaats van constanten mogen ook strings worden ingevoerd. Hierop zal pas in hoofdstuk 10 worden ingegaan.
In het programma wordt één datagebied gemaakt dat alle waarden die door DATA statements worden aangegeven, achter elkaar neerzet.
Een DATA statement moet het enige statement op een regel zijn. Er mogen dan niet meer statements op die regel voorkomen.

Foutief is dus bijvoorbeeld:

```
*10 LET A=5:DATA 100,200,300
10    *** LET A=5:DATA? 100,200,300
SYNTAX ERROR
*
```

evenals:

```
*10 DATA 100,200,300:LET A=5
*
```

Hier zal 300: LET A = 5 als string worden opgevat, zoals in hoofdstuk 10 nader wordt uitgelegd. De toekenning 'A wordt 5' treedt niet op.

Let wel! Alleen constanten mogen in een DATA statement voorkomen, expressies niet.
Goed is dus:

```
*LIST
10     DATA 100,200,300
*
```

maar fout is:

```
*10 DATA 100,100+100,300
*
```

Ook hier wordt weer een stringinterpretatie toegepast en niet de bedoelde optelling.
Alle DATA statements vormen samen maar één datablok dat gevuld is met gegevens.

Voorbeeld

```
*LIST
10     DATA 5,36,84
20     DATA 100,200,300
*
```

heeft exact hetzelfde effect als:

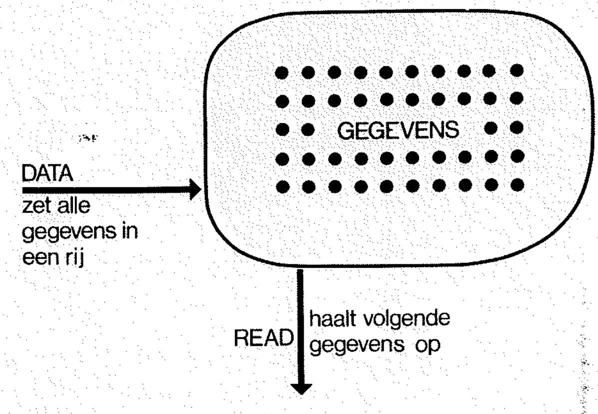
```
*LIST
10     DATA 5,36,84,100,200,300
*
```

namelijk een datalijst met de waarden 5, 36, 84, 100, 200, 300.

De gegevens die op deze manier in een lijst worden opgeborgen, vormen als het ware een onderdeel van het programma en kunnen tijdens de executie ervan niet veranderen. Hoe vaak het programma ook wordt uitgevoerd, toch zijn de waarden, die worden opgeborgen, steeds dezelfde. Dit in tegenstelling tot het INPUT statement. Bij het gebruik van een INPUT statement moet de waarde worden ingetypt en kan dus naar wens van de typist(e) veranderen.

7.3. Het statement READ

Wanneer we nu de gegevens uit de lijst in het programma willen lezen, staat in BASIC daarvoor het statement READ ter beschikking. READ leest telkens één waarde uit de met DATA statements ge-



Afb. 7.2. Samenwerking van DATA en READ

reedstaande lijst (zie afbeelding 7.2) en kent die toe aan een variabele, die achter het keyword READ gespecificeerd is. Voor elke variabele wordt een volgende waarde uit de lijst gehaald.

Voorbeeld

```
*LIST
10    DATA 100,200,300
20    READ A
30    READ B
40    READ C
50    PRINT A,B,C
*RUN
 100.0        200.0        300.0
*
```

Op regel 10 wordt de lijst gevuld. Op regel 20 wordt de eerste waarde uit de lijst gehaald en aan de variabele A toegekend. Na uitvoering van regel 20 heeft A dus de waarde 100 gekregen.
We merken daarbij op dat de meest linkse waarde de eerste waarde in het datagebied is. Na uitvoering van regel 30 heeft B de waarde 200 gekregen. Na regel 40 heeft C de waarde 300.
In dit voorbeeld is het DATA statement eerst gegeven en daarna READ. BASIC staat evenwel ook toe dat de READ statements voorafgegaan aan het DATA statement. Bij de aanroep van het programma zet BASIC de waarden als het ware al in het datagebied. Wanneer we het programma vervolgens met behulp van het RUN commando laten executeren, staan de gegevens al klaar.

```
*LIST
10    READ A
20    READ B
30    PRINT A,B
40    DATA 100,200
*RUN
 100.0        200.0
*
```

Vele BASIC programmeurs hebben de gewoonte de DATA statements aan het eind van hun programma te zetten. Dit vergemakkelijkt het *terugvinden* van de vulling van het DATA gebied. Uiteraard gebeurt dit ook omdat de programmeur aan het eind van zijn programma zeker bekend is met de waarden, die hij in het DATA gebied wenst op te slaan.
Het is niet nodig dat in een READ statement slechts één variabele één waarde krijgt toegekend. Ook meerdere variabelen mogen worden gebruikt, mits ze door komma's zijn gescheiden.
Het in bovengenoemd voorbeeld verkregen resultaat zal dan ook kunnen worden bereikt door:

```
*LIST
10    DATA 100,200,300
20    READ A,B,C
30    PRINT A,B,C
*RUN
 100.0        200.0        300.0
```

Ook nu krijgt A de waarde 100, B de waarde 200 en C de waarde 300 na de executie van regel 20.

Het programma

```
*LIST
10     READ A,B,C
20     DATA 100,200,300
30     PRINT A,B,C
*RUN
 100.0        200.0        300.0
*
```

geeft uiteraard hetzelfde effect.
Tenslotte moet nog worden opgemerkt dat dezelfde variabele in verschillende READ statements een waarde mag krijgen. Dit blijkt uit het volgende programma:

```
*LIST
10     READ A,B
20     PRINT A,B
30     READ A
40     PRINT A
50     DATA 10,20,30
*RUN
 10.0         20.0
 30.0
*
```

Statement	Effect
READ	Inlezen van gegevens uit een DATA gebied.

Algemene vorm:
regelnummer READ v_1 [, v_2, ..., v_n]

waarbij:

v_1, ..., v_n de namen van variabelen zijn.

Dus na het regelnummer mogen er een of meer blanco's voorkomen, gevolgd door READ. Daarna volgen een of meer blanco's en vervolgens tenminste één variabele naam. Wanneer meer variabele namen voorkomen, moeten deze door komma's zijn gescheiden. Achter de laatste variabele naam mag geen komma staan.

Effect:
De met een DATA statement in het datagebied klaargezette gegevens worden aan de in READ genoemde variabelen toegekend. Het eerst geëxecuteerde READ statement start bij het ophalen met de eerste waarde uit het datagebied, dat wil zeggen de waarde die links in het eerste DATA statement staat. Daarna wordt de volgende waarde uit dit DATA statement gelezen, enz.

Wanneer de waarden, die door het eerste DATA statement zijn neergezet, verbruikt zijn, worden de waarden uit het volgende DATA statement opgehaald. Bij het gebruik van het DATA statement en het READ statement moeten we voor een correct resultaat de volgende regels in acht nemen:

a. In de eerste plaats moeten de DATA en READ statements correct gespecificeerd zijn, zoals hun algemene vorm weergeeft.
b. De waarden en het soort van de gegevens in de via DATA statements gevulde lijst moeten in volgorde en in betekenis overeengekomen met de waarden en het soort, die de READ statements veronderstellen. Wanneer we bijvoorbeeld het eerste getal gebroken willen laten zijn, moet dat ook inderdaad een gebroken getal zijn.
   In hoofdstuk 10 gaan we in op tekstgebruik in BASIC. We zullen dan zien dat we via een DATA en een READ statement ook tekst kunnen invoeren. Tekst mag natuurlijk niet met getallen worden verwisseld.
c. Er moeten in het DATA statement tenminste evenveel gegevens in de lijst worden gezet als er variabelen in de READ statements zijn. Als er meer gegevens in de lijst staan dan in het READ statement worden opgehaald, worden de niet opgehaalde gegevens gewoon niet gebruikt. Wanneer we meer gegevens uit de lijst proberen te halen met behulp van de READ statements dan er met de DATA statements zijn ingezet, zal onze computer een foutmelding geven.

Voorbeeld

```
*LIST
10    DATA 100,200,300
20    DATA 400,500,600,700
30    READ A,B,C,D
40    PRINT A,B,C,D
*RUN
 100.0         200.0         300.0         400.0
*
```

Dit is correct. Na uitvoering zijn de waarden voor A, B, C, en D respectievelijk 100, 200, 300, 400. De waarden 500, 600 en 700 worden hier dus niet gebruikt. We kunnen ons dit voorstellen zoals is weergegeven in afbeelding 7.3.

datablok	*gebruikt*
100	nee
200	nee
300	nee
400	nee
500	nee
600	nee
700	nee

situatie voor READ A, B, C, D

datablok	*gebruikt*
100	ja
200	ja
300	ja
400	ja
500	nee
600	nee
700	nee

situatie na READ A, B, C, D

Afb. 7.3. Gebruik van een datablok

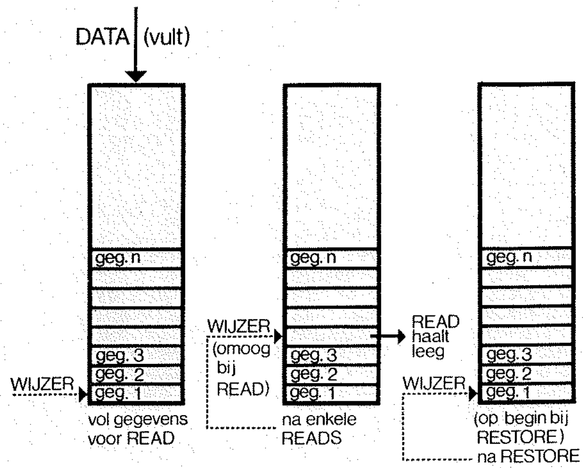
Ander voorbeeld

```
*LIST
10    DATA 100,200,300
20    DATA 400,500,600,700
30    READ A,B,C,D,E,F,G,H,I
*RUN
OUT OF DATA IN LINE 30
*
```

Dit is niet correct.
We zien dan ook de foutmelding 'OUT OF DATA IN LINE 30' verschijnen. We proberen hier namelijk 9 gegevens uit de lijst te lezen, terwijl we er slechts 7 in hebben gezet.

7.4. Het RESTORE statement

In de computer wordt intern bijgehouden welke gegevens uit de lijst al zijn gebruikt We kunnen dit vergelijken met het zetten van een kruisje op een lijst, die we op papier hebben opgesteld. Soms is het wenselijk weer boven aan de lijst te kunnen beginnen. Hiervoor kennen we in BASIC de functie RESTORE (zie afbeelding 7.4).
Het eerste READ statement na RESTORE zal dus weer gegevens gaan halen uit het eerste DATA statement.



Afb. 7.4. DATA/READ/RESTORE

Als we RESTORE niet gebruiken zien we het resultaat in het onderstaand programma.

```
*LIST
10      DATA 100,200,300
20      READ A,B,C
30      DATA 400
40      READ D
50      PRINT D
*RUN
 400.0
*
```

Na executie heeft D de waarde 400.
Is het programma echter als volgt van RESTORE voorzien:

```
*LIST
10      DATA 100,200,300
20      READ A,B,C
30      DATA 400
35      RESTORE
40      READ D
50      PRINT D
*RUN
 100.0
*
```

dan heeft D na executie de waarde 100 gekregen. Hierbij wordt verwezen naar afbeelding 7.5.

datablok	*gebruikt*
100	nee
200	nee
300	nee

situatie voor READ A, B, C

datablok	*gebruikt*
100	ja
200	ja
300	ja

situatie na READ A, B, C

datablok	*gebruikt*
100	ja
200	ja
300	ja
400	nee

situatie voor RESTORE

datablok	*gebruikt*
100	nee
200	nee
300	nee
400	nee

situatie na RESTORE

datablok	*gebruikt*
100	ja
200	nee
300	nee
400	nee

situatie na READ D

Afb. 7.5. Gebruik van RESTORE

Statement	Effect
RESTORE	Lezen zal weer van voren af aan in het datagebied gebeuren.

Algemene vorm:

regelnummer RESTORE

Er is dus geen enkele toevoeging aan RESTORE nodig.

Effect:

Een READ statement zal de gegevens weer van voren af aan uit het datagebied lezen. (De wijzer (pointer) wordt op het begin gezet.)

N.B.

END statement mag worden weggelaten.

Voorbeeld

```
*LIST
10    DATA 2,4,6,8,10,12,14
30    READ J,K,L
35    PRINT J,K,L
50    READ M,N,P,Q
55    PRINT M,N,P,Q
80    RESTORE
90    READ R,S
95    PRINT R,S
*RUN
 2.0          4.0          6.0
 8.0          10.0         12.0         14.0
 2.0          4.0
*
```

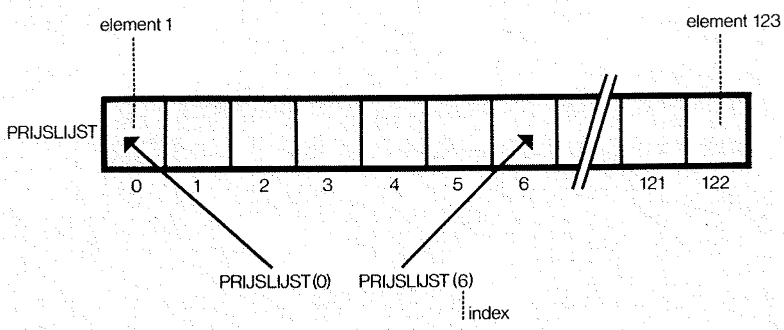
In dit voorbeeld hebben J, K en L dus respectievelijk de waarden 2, 4 en 6 gekregen M, N, P en O hebben de waarden 2, 4, 6 en 8, terwijl R en S de waarden 2 en 4 kregen.

```
*LIST
10    DATA 2,4,6,8,10,12,14
30    READ J,K,L
35    PRINT J,K,L
40    RESTORE
50    READ M,N,P,Q
55    PRINT M,N,P,Q
80    RESTORE
90    READ R,S
95    PRINT R,S
*RUN
 2.0          4.0          6.0
 2.0          4.0          6.0          8.0
 2.0          4.0
*
```

Nu hebben M, N, P en O de waarden 8, 10, 12 en 14 gekregen. Terwijl J, K, L, R en S dezelfde waarde hebben als in het voorgaande voorbeeld.
Een andere manier waarop we uiteraard de gegevens weer vooraan uit de lijst kunnen halen, is het programma opnieuw te laten executeren met behulp van het commando RUN. Dan begint het programma gewoon opnieuw van voren af aan.

7.5. Het statement DIM

Het is dikwijls wenselijk om in een programma een aantal waarden onder één en dezelfde naam aan te kunnen roepen.
Als voorbeeld kan een *prijslijst* dienen. Daarin kunnen de prijzen van verschillende artikelen worden samengevat onder de naam P. Stel dat we te maken hebben met 123 verschillende artikelen. De prijslijst moet dan 123 elementen lang zijn. Als we de prijs van een bepaald artikel willen weten, moeten we het nummer van het artikel opgeven.



Afb. 7.6. Prijstabel array

Het zevende artikel in de lijst kunnen we dan opvragen door middel van de uitdrukkingen P(6). We nummeren in BASIC namelijk vanaf 0! (zie afbeelding 7.6).
De prijslijst van dit voorbeeld heet in BASIC termen een 'array'. Een array is dus een lijst van waarden, die we onder één naam kunnen aanroepen. Het nummer achter de prijslijst dat tussen haakjes wordt geplaatst, heet in BASIC termen 'subscript' *(index).*
Het spreekt vanzelf dat de computer niet zonder meer weet hoeveel waarden we in een 'array' willen opnemen. Toch moet de computer voldoende ruimte voor al deze waarden scheppen. In een BASIC programma hebben we daarvoor de beschikking over het DIM statement. DIM staat voor dimension (dimensie) en geeft de lengte van de gebruikte array weer.
Om onze prijslijst aan te geven gebruiken we dan ook het statement:

```
*LIST
110   DIM P(122.0)
*
```

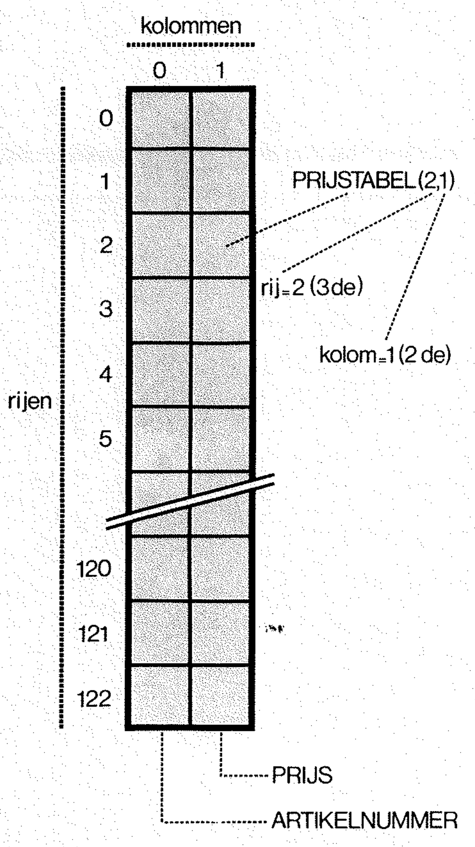
Hiermee geven we de computer te kennen dat hij voor onze array P een ruimte moet maken van 123 plaatsen, genummerd van 0 tot en met 122. In feite geven we dus in DIM de hoogste subscriptwaarde op. De Teleac/DAI personal computer heeft de onhebbelijke

eigenschap alle waarden als floating-point getallen uit te printen. Ingetypt was DIM P(122). Bij listen volgt DIM P(122.0). Dit is correct, maar niet elegant. We zien dit verschijnsel in alle hoofdstukken terug. Over de waarde van deze getallen wordt in het DIM statement nog niets gezegd. Het systeem vult daarom de waarde 0 in voor iedere plaats in de matrix. Indien we andere waarden wensen, moeten we verderop in het programma daarvoor door middel van toekenningen maatregelen treffen.
De array zoals we die nu hier zagen, bestaat uit een lange rij getallen. Deze rij kent slechts één dimensie, namelijk zijn lengte. In de wiskunde zegt men gewoonlijk vector in plaats van array voor zo'n geval. BASIC gebruikt steeds de naam array. Stel dat we nu een tabel willen maken van getallen. We zetten dan rijen getallen onder elkaar. Dit doen we voor een prijslijst, waarin artikelnummers en daarnaast hun prijzen staan. Wanneer we over 123 verschillende artikelnummers beschikken, schrijven we 123 verschillende regels op. Iedere regel bestaat uit een rij van 2 getallen. Het eerste getal is het artikelnummer, het tweede getal de bijbehorende prijs. In de wiskunde spreekt men dan ook over een array.
Wanneer we deze tabel de naam T geven en tegelijkertijd voldoende ruimte reserveren, kan dat door middel van het volgende statement:

```
*LIST
10     DIM T(122.0,1.0)
*
```

De array T bestaat uit 123 regels van 2 getallen. Voor elk van die getallen is nu ruimte gereserveerd. Wanneer we van de derde regel het tweede getal willen hebben, kunnen we dat krijgen door middel van de uitdrukking: T(2,1). Op dezelfde manier kunnen we ook de waarden in de tabel T(2,1) aanbrengen (zie afbeelding 7.7).
Als voorbeeld geven we het vullen van de 12e regel, het tweede element:

```
*LIST
10     DIM T(112.0,1.0)
20     LET T(11.0,1.0)=15.0
30     PRINT T(11.0,0.0),T(11.0,1.0)
*RUN
 0.0          15.0
*
```



Afb. 7.7. Array met twee dimensies

De correcte waarde op de 12e regel in tabel T is nu 15 geworden en de andere getallen hebben nog steeds de waarde 0. Als voorbeeld kiezen we het volgende programma:

```
*LIST
10     REM VOORBEELDPROGRAMMA ARRAY
20     DIM A(6.0,10.0)
30     FOR I=0.0 TO 6.0
40     LET A(I,0.0)=I
50     FOR J=0.0 TO 10.0
60     LET A(0.0,J)=J
```

```
70    PRINT A(I,J);" ";
80    NEXT J
90    PRINT
100   NEXT I
110   END
*RUN
 0.0  1.0  2.0  3.0  4.0  5.0  6.0  7.0  8.0  9.0  10.0
 1.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
 2.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
 3.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
 4.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
 5.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
 6.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
END PROGRAM
*
```

In regel 20 geven we op dat de array A uit 7 regels van 11 getallen bestaat. Regels 50 tot en met 80 vullen de eerste regel met de getallen 0 tot en met 10 en print deze op het scherm. De regels 30 tot en met 100 zetten daarop als eerste getal van iedere regel de getallen 0 tot en met 6. Regel 90 zorgt er daarbij voor dat er steeds aan een nieuwe regel wordt begonnen, nadat de vorige is uitgeprint. In BASIC zijn we op de Teleac/DAI personal computer niet beperkt tot maximaal 2 dimensies.

Voorbeeld

```
*LIST
10    DIM A(3.0,2.0,7.0,4.0,8.0,9.0)
*
```

Hier geven we een 6-dimensionale array A op. De dimensie bevat daarbij 4 getallen (tellend van 0 tot en met 3). De tweede dimensie 3 getallen, enz. We mogen in een DIM statement ook de dimensies opgeven van meer arrays.
Dus ook is toegestaan:

```
*LIST
10    DIM A(3.0,4.0,5.0),N(5.0),W(6.0,7.0,8.0)
*
```

Hierbij hebben we ruimte gereserveerd voor een array A met 3 dimensies, een array N met 1 dimensie en een array W met 3 dimensies. Bij verstandig gebruik van het DIM statement moeten we het aantal getallen in een array niet te groot maken. De computer heeft een beperkte geheugenruimte tot zijn beschikking. Als we arrays onnodig groot maken, verliezen we daarin een groot deel van de ons ter beschikking staande ruimte. De array W kost ons 7 × 8 × 9 = 504 getallen.

Statement	Effect
DIM	Declareer de dimensies van een array.

Algemene vorm:

regelnummer DIM v_1 (s_{11} [, s_{12}, ..., s_{1n}]) [, v_2(s_{21} [, s_{22} ..., s_{2n}]), ...]

waarbij $v_1 \ldots v_n$ namen van variabelen zijn, s numerieke constanten, numerieke variabelen of rekenkundige expressies zijn.

Effect:

De variabelen v_1 tot en met v_n worden als array gedeclareerd. De dimensies worden opgegeven volgens de waarden, die door s worden aangegeven.
De dimensies van de variabelen worden opgegeven door middel van het zogenaamde subscript.

In verschillende versies van BASIC is het overbodig om 1-dimensionale arrays met een lengte van 10 of kleiner met behulp van DIM eerst te declareren. Het systeem neemt voor iedere variabele die als array wordt gebruikt aan dat het een array van een lengte 10 is. Wanneer gebruik wordt gemaakt van die array zonder subscript, veronderstelt het systeem dat de variabele met subscriptwaarde 0 wordt bedoeld.
Er bestaan ook BASIC versies, waarin zelfs een 2-dimensionale array tot 10 × 10 niet met DIM gedeclareerd hoeft te worden. In de Teleac/DAI personal computer moet altijd het DIM statement worden gebruikt. Dit is in de programmering overigens ook wel zo duidelijk: men ziet direct in het programma hoe groot een array is.
Voor dit soort verschillen moet men de handleiding van de eigen computer raadplegen.
Het is nu ook duidelijk dat de algemene manier, waarop we naar getallen in een array kunnen referen door middel van subscripts, de volgende vorm heeft:

variabele (subscript [, subscript...])

Het aantal subscripts moet even groot zijn als het aantal dimensies, dat de array in het DIM statement verkregen heeft.

Een correct voorbeeld is dus

```
*LIST
10     DIM A(2.0,3.0,4.0)
11     PRINT A(0.0,2.0,1.0)
*RUN
 0.0
*
```

Er treedt een fout op als we dat niet doen.

Voorbeeld

```
*LIST
10    DIM A(2.0,3.0,4.0)
20    PRINT A(0.0,2.0)
*RUN
SUBSCRIPT ERROR IN LINE 20
*
```

Het is hier onduidelijk welk getal uit de array we nu bedoelen. Evenzo gaat het mis wanneer we meer subscripts proberen te gebruiken.

Dus als voorbeeld

```
*LIST
10    DIM A(2.0,3.0,4.0)
20    PRINT A(0.0,2.0,1.0,3.0)
*RUN
SUBSCRIPT ERROR IN LINE 20
*
```

Ook aan de subscript zelf worden eisen gesteld. De waarde daarvan moet groter of gelijk zijn aan nul, maar kleiner of gelijk aan de dimensie, waarop de subscript betrekking heeft. Correcte voorbeelden hebben we hierboven reeds gezien.

Een ander correct voorbeeld

```
*LIST
10    DIM A(2.0,3.0,4.0)
20    LET I=1.0
30    PRINT A(I-1.0,I,I*2.0)
*RUN
 0.0
*
```

Niet correct is

```
*LIST
10    DIM A(2.0,3.0,4.0)
20    PRINT A(-1.0,2.0,3.0)
*RUN
NUMBER OUT OF RANGE IN LINE 20
*
```

want −1 is hier niet toegestaan.

Ook niet correct is

```
*LIST
10    DIM A(2.0,3.0,4.0)
20    PRINT A(4.0,2.0,1.0)
*RUN
SUBSCRIPT ERROR IN LINE 20
*
```

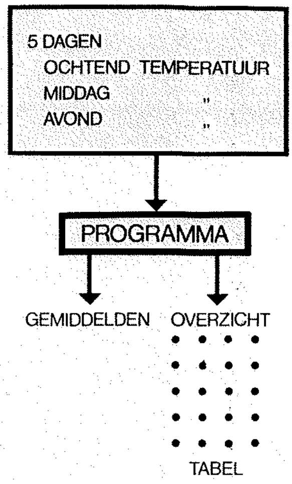
want de eerste dimensie in regel 10 is kleiner dan de subscript 4, die we in regel 20 aantreffen.
Wanneer een waarde als subscript wordt gebruikt, maakt de Teleac/DAI personal computer er automatisch een geheel getal (integer) van. Dit gebeurt door het weglaten van de cijfers achter de komma, zoals uit onderstaande voorbeelden blijkt. Dit geldt algemeen voor de Teleac/DAI personal computer. Wanneer in BASIC een gebroken getal wordt gebruikt, daar waar een geheel getal nodig is, maakt de computer er een geheel getal van door afkappen (ofwel afronden naar beneden voor een positief getal).

```
*LIST
10    DIM A(9.0)
20    FOR I=0.0 TO 9.0
30    A(I)=I
40    PRINT A(I);
50    NEXT I
60    PRINT
70    PRINT A(0.0);A(0.1);A(0.99);A(1.0);A(1.01);A(2.0)
80    END
*RUN
 0.0 1.0 2.0 3.0 4.0 5.0 6.0 7.0 8.0 9.0
 0.0 0.0 0.0 1.0 1.0 2.0
END PROGRAM
*
```

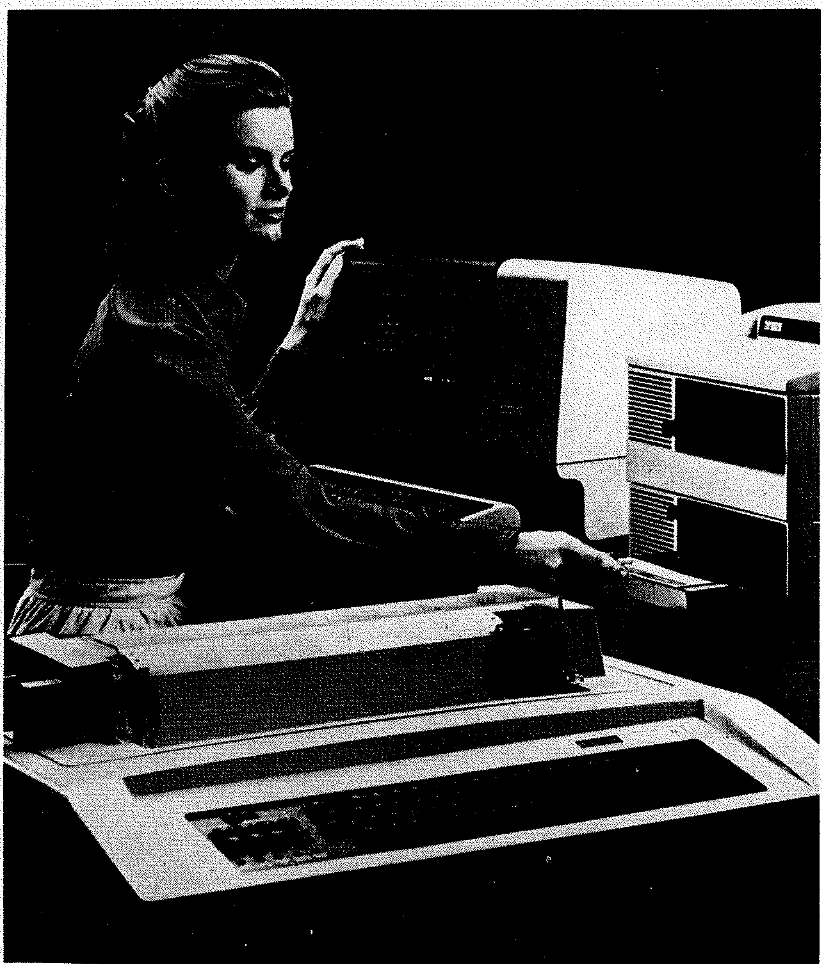
Door de beperkte rekennauwkeurigheid van BASIC in gebroken getallen kunnen hierdoor wel eens onverwachte subscripts ontstaan. Wanneer men zekerheid wil hebben of de subscript uit een berekening met gebroken getallen ontstaat, kan de subscript beter zelf eerst tot het gewenste gehele getal (integer) worden afgerond. Voor een positief gebroken getal G kan dit bijvoorbeeld door de uitdrukking INT (G + 0.5) worden bereikt zoals in onderstaand voorbeeld (zie hoofdstuk 8).

```
*LIST
10    FOR G=0.0 TO 1.0 STEP 0.1
20    PRINT G,INT(G+0.5)
30    NEXT G
*RUN
 0.0          0.0
 0.1          0.0
 0.2          0.0
 0.3          0.0
 0.4          0.0
 0.5          1.0
 0.6          1.0
 0.7          1.0
 0.8          1.0
 0.9          1.0
 1.0          1.0
*
```

Het valt hier uiteraard ook op dat een 'gerund' (uitgevoerd) programma wordt afgesloten met de mededeling END PROGRAM, indien het laatste statement een END was. Is dit niet het geval, dan blijft deze melding achterwege.
Ook voor tekst in BASIC is het mogelijk arrays op te bouwen. Zij bevatten dan *karakters* in plaats van *getallen*. Hoe dit gaat, wordt in hoofdstuk 10 behandeld.
Als voorbeeld van een programma dat met succes arrays gebruikt, kijken we nu naar een programma dat voor 5 dagen de temperatuur 's morgens, 's middags en 's avonds van een dag uitprint. Er wordt een overzicht geproduceerd over 5 dagen met hun *gemiddelde* temperatuur (zie afbeelding 7.8).



Afb. 7.8. Een tabel via een array



```
LIST
10    REM PROGRAMMA VOOR GEMIDDELDE DAGTEMPERATUUR
20    DIM T(5.0,3.0)
30    REM LEES DE DATA NU IN
40    FOR D=0.0 TO 4.0
50    FOR M=0.0 TO 2.0
60    READ T(D,M)
70    NEXT M
80    NEXT D
90    REM BEREKEN HET DAGGEMIDDELDE
100   FOR D=0.0 TO 4.0
110   LET H=0.0
120   FOR M=0.0 TO 2.0
130   LET H=H+T(D,M)
140   NEXT M
150   LET T(D,3.0)=H/3.0
160   NEXT D
170   REM BEREKEN HET GEMIDDELDE OVER DE OCHTEND,MIDDAG. EN
180   FOR M=0.0 TO 3.0
190   LET H=0.0
200   FOR D=0.0 TO 4.0
210   LET H=H+T(D,M)
220   NEXT D
230   LET T(5.0,M)=H/5.0
240   NEXT M
250   REM NU   UITPRINTEN
260   PRINT " ","MORGEN","MIDDAG","AVOND","GEM."
270   PRINT
280   FOR D=0.0 TO 4.0
290   PRINT "DAG";D,
300   FOR M=0.0 TO 3.0
310   PRINT T(D,M),
320   NEXT M
325   PRINT
330   NEXT D
340   PRINT
350   PRINT "GEMIDDELD",
360   FOR M=0.0 TO 3.0
370   PRINT T(5.0,M),
380   NEXT M
390   PRINT
400   DATA 10,12,9,12,15,13,12,13,12
410   DATA 11,11,11,9,10,8
420   END
*RUN
             MORGEN      MIDDAG      AVOND       GEM.

DAG 0.0      10.0        12.0        9.0         10.3333
DAG 1.0      12.0        15.0        13.0        13.3333
DAG 2.0      12.0        13.0        12.0        12.3333
DAG 3.0      11.0        11.0        11.0        11.0
DAG 4.0      9.0         10.0        8.0         9.0

GEMIDDELD    10.8        12.2        10.6        11.2

END PROGRAM
*
```

In regel 20 wordt de array T, die de temperaturen bevat, geïnitieerd als 6 regels met ieder 4 waarden. Zo'n regel zal de 3 temperaturen op die dag en de gemiddelde dagtemperatuur bevatten.
Regels 50 tot en met 70 lezen de temperaturen uit het DATA gebied in de array T. dit doen we met een READ statement uit een DATA gebied, zodat we ook de combinatie DATA en READ nog eens zien. De variabele D wordt steeds gebruikt voor de dagindex en M zullen we gebruiken voor het moment van de dag zoals 's morgens, enz.
In regels 100 tot en met 160 bepalen we de daggemiddelden. De variabele H fungeert als hulpvariabele. Per dag wordt het daggemiddelde bepaald op de regels 120 tot en met 160. We tellen dan de temperaturen van 1 dag 's morgens, 's middags en 's avonds op en delen dit totaal door 3 om het gemiddelde te bepalen (regel 150).
De regels 180 tot en met 240 berekenen het gemiddelde van de verschillende dagmomenten over de 5 dagen, die we hier hebben ingevuld.
Regels 260 en 270 printen een kop uit, waarna op de regels 280 tot en met 330 de verschillende dagwaarden worden geprint. Het printen van de waarden van één dag vindt plaats in de regels 300 tot en met 320. We zien daar dat we de mogelijkheid van het PRINT statement benutten om na het printen op dezelfde regel te blijven. Daarom is het PRINT statement met een komma (,) afgesloten. De regels 360 tot en met 380 printen het gemiddelde van dagmomenten uit. Deze te printen regels worden voorafgegaan en gevolgd door een blanco regel.
In regels 400 en 410 vullen we de gemeten dagtemperaturen in het DATA gebied in. De DATA statements staan hier dus onderaan het programma, wat toegestaan en veelal gebruikelijk is.

7.6. SAVEA en LOADA

In hoofdstuk 5 hebben we gezien dat het door middel van SAVE en LOAD mogelijk is om programma's op cassetteband te zetten en weer te laden, terwijl daar in hoofdstuk 19 nog eens op teruggekomen wordt. Evenzo is het uiteraard wenselijk om gewone gegevens op cassette weg te schrijven en opnieuw te laden. Denk maar eens aan administratieve programma's, waarin bijvoorbeeld de lijst van klanten wordt opgeborgen en weer herladen. Voor dit doel zijn de commando's SAVEA en LOADA in BASIC gedefinieerd. Door middel van SAVEA wordt een variabele naar cassette weggeschreven. We kunnen weer een naam aan de weggeschreven informatie toekennen, net als bij een programma. SAVEA dient de volgende vorm te hebben:

```
SAVEA VAR "NAAM"
```

Hierin is VAR de naam van een variabele, terwijl NAAM de naam is waaronder deze variabele op de cassette wordt weggeschreven. Stel dat we de variabele K willen wegschrijven onder de naam KLANTEN, dan wordt de aanroep:

```
SAVEA K "KLANTEN"
```

Wanneer we later deze variabele weer wensen in te lezen zeggen we eenvoudig:

```
LOADA K "KLANTEN"
```

Na afloop heeft de variabele K de vroeger onder KLANTEN weggeschreven waarde(n) weer verkregen. Uiteraard mogen we deze waarden ook aan een andere variabele toekennen, bijv.:

LOADA N "KLANTEN"

Na afloop heeft nu N bovengenoemde waarde(n) aangenomen. Wat er gelezen wordt is namelijk bepaald door de naam KLANTEN die mede weggeschreven is op de cassette. De naam van de variabele, die weggeschreven is (in ons voorbeeld K) doet bij het teruglezen in het geheel niet meer mee. De vorm die LOADA dient te hebben is dus:

LOADA VAR "NAAM"

waarin VAR weer een naam van een variabele is, terwijl NAAM de naam is waaronder de waarden die VAR nu verkrijgt vroeger op de cassetteband zijn weggeschreven.
SAVEA en LOADA mogen ook als statement in een programma verschijnen. Op deze wijze wordt het dus mogelijk onder besturing van een BASIC programma gegevens naar cassette te schrijven en terug te lezen. Aangezien de variabele een array is, wordt hier een grote mate van flexibiliteit verkregen. De in hoofdstuk 10 te beschrijven array, bestaande uit louter tekst, is hierbij ook toegestaan. Voorwaarde bij het gebruik van de arrayvariabelen is dat het aantal elementen van de weggeschreven array minstens even groot is als het aantal terug te lezen elementen. Zou de computer meer elementen moeten lezen dan er op de cassette staan dan geeft hij een foutmelding. Staan er teveel elementen op de band dan worden alleen die elementen gebruikt die echt nodig zijn, de rest blijft ongebruikt. Het spreekt vanzelf dat ook het type van de weggeschreven variabele (integer, floating-point, enz.) hetzelfde moet zijn als van de in te lezen variabele.
In feite geldt bij SAVEA/LOADA hetzelfde als bij de combinatie DATA/READ, die we in dit hoofdstuk reeds zagen.
Lees evenwel weer aandachtig de gebruiksaanwijziging van uw computer, want ook hier kunnen weer kleine verschillen optreden.

Samenvatting hoofdstuk 7

In dit hoofdstuk hebben we gezien hoe we gegevens kunnen opslaan met behulp van het DATA statement en terug kunnen lezen met READ. Verder is getoond hoe via RESTORE het weer te lezen is vanaf het begin. Daarna zijn we ingegaan op het begrip 'array'. Een array wordt eerst gedeclareerd via een DIM statement. Een waarde uit zo'n array kunnen we terugvinden met behulp van subscripts. Ten slotte is getoond hoe een array naar een band geschreven kan worden en kan worden geladen vanaf een band.

Multiple choice-vragen bij hoofdstuk 7

A. De READ en DATA statements zorgen er samen voor dat:

1. gegevens op cassetteband opgeslagen en daarna gelezen en toegekend worden aan variabelen
2. gegevens in het geheugen van de computer worden opgeslagen en op het beeldscherm afgedrukt
3. gegevens in het geheugen van de computer opgeslagen en daarna gelezen en toegekend worden aan variabelen
4. de waarden van de variabelen in het geheugen van de computer worden opgeslagen

B. **a.** in een programma mogen meer READ dan DATA statements voorkomen.
b. In een programma mogen meer DATA dan READ statements voorkomen.

1. uitspraak a is goed en uitspraak b is goed
2. uitspraak a is fout en uitspraak b is goed
3. uitspraak a is goed en uitspraak b is fout
4. uitspraak a is fout en uitspraak b is fout

C. In DATA statements mogen geen:

1. constanten voorkomen, die een geheel getal weergeven
2. constanten voorkomen, die een gebroken getal weergeven
3. expressies voorkomen
4. strings voorkomen

D. **a.** READ statements moeten worden voorafgegaan door DATA statements.
b. READ statements moeten worden gevolgd door DATA statements.

1. uitspraak a is goed en uitspraak b is goed
2. uitspraak a is fout en uitspraak b is goed
3. uitspraak a is goed en uitspraak b is fout
4. uitspraak a is fout en uitspraak b is fout

E. **a.** In een READ statement mogen meer variabelen een waarde krijgen.
b. Dezelfde variabele mag niet in meer READ statements een waarde krijgen.

1. uitspraak a is goed en uitspraak b is goed
2. uitspraak a is fout en uitspraak b is goed
3. uitspraak a is goed en uitspraak b is fout
4. uitspraak a is fout en uitspraak b is fout

F. Het doel van het RESTORE statement is:

1. het geheugen schoonvegen
2. de waarden van de ingelezen variabelen herstellen
3. het in het geheugen aanwezige programma herstarten
4. de volledige DATA-lijst weer ter beschikking te stellen

G. Een DIM statement moet in een BASIC programma voorkomen als:

1 numerieke variabelen voorkomen
2 tabellen worden uitgeprint
3 haakjes () zijn gebruikt
4 array's voorkomen

H. a. Het aantal subscripts in een array mag kleiner zijn dan het aantal dimensies van het array.
b. De waarde van een subscript in een array moet kleiner of gelijk zijn aan het getal dat de overeenkomstige dimensie van het array voorstelt.

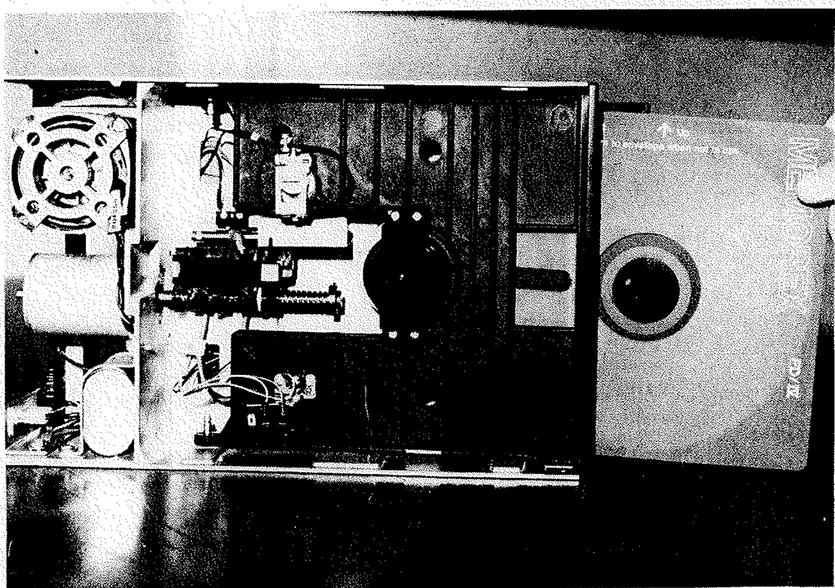
1 uitspraak a is goed en uitspraak b is goed
2 uitspraak a is fout en uitspraak b is goed
3 uitspraak a is goed en uitspraak b is fout
4 uitspraak a is fout en uitspraak b is fout

I. a. Er mogen meer DIM statements op een regel voorkomen.
b. In een DIM statement mogen rekenkundige expressies worden gebruikt.

1 uitspraak a is goed en uitspraak b is goed
2 uitspraak a is fout en uitspraak b is goed
3 uitspraak a is goed en uitspraak b is fout
4 uitspraak a is fout en uitspraak b is fout

J. a. DIM A(10, 8) declareert een array A van 80 elementen.
b. Het hoogste subscript in het gedeclareerde array A van uitspraak a is 10, het laagste is 0.

1 uitspraak a is goed en uitspraak b is goed
2 uitspraak a is fout en uitspraak b is goed
3 uitspraak a is goed en uitspraak b is fout
4 uitspraak a is fout en uitspraak b is fout



Afb. 7.9. Het mechanische deel van een floppy-disk drive

Hoofdstuk 8

BASIC functies

In dit hoofdstuk zullen we ingaan op het begrip functie. Een groot aantal functies, die in de Teleac/DAI personal computer voorkomen, worden behandeld.

8.1. Inleiding

In hoofdstuk 6 is het gebruik van *subroutines* uitgelegd. We zagen dat het verstandig is delen van een programma, die algemeen kunnen worden gebruikt, als subroutines te programmeren. Zo'n subroutine kan dan vanuit een programma worden aangeroepen met een GOSUB statement.
Iedere subroutine moet met een RETURN worden afgesloten. Het aanroepende programma wordt daarna op de regel na de GOSUB aanroep voortgezet.
Het is duidelijk dat er subroutines kunnen worden gemaakt, die voor een groot aantal gebruikers van het computersysteem bruikbaar zijn. Nog prettiger is het wanneer we deze niet meer als subroutine hoeven aan te roepen, maar gewoon onder een duidelijke naam. Daarom zijn de zogenaamde functies ontstaan.

8.2. Het begrip 'functie'

In BASIC staan een aantal van deze routines ter beschikking. Ze worden dan niet subroutines genoemd, maar functies. Deze functies zijn dus door de leverancier van de Teleac/DAI personal computer reeds voorgeprogrammeerd.

In de Engelstalige literatuur wordt daarvoor dikwijls de naam *intrinsic function* (= innerlijke functie, dat wil zeggen in het systeem aanwezige functie) gebruikt. Soms gebruikt men ook de naam *library function* (= bibliotheek-functie). In dit boek worden ze steeds functie genoemd.

Het verschil met een subroutine is dus in de eerste plaats dat de gebruiker zelf de routine niet meer hoeft te programmeren, in de tweede plaats worden de functies met behulp van een naam aangeroepen.
Een voorbeeld van een functie is ABS(X). Deze functie geeft de absolute waarde van een getal, dat wil zeggen de waarde van het getal zonder daarbij op het teken (+ of −) te letten.

Voorbeeld

```
*LIST
10      LET A=ABS(3.21)
20      PRINT A
30      PRINT ABS(-3.21)
40      END
*RUN
 3.21
 3.21
END PROGRAM
*
```

We zien hier dus dat ABS(X) de waarde van het argument behoudt zonder dat rekening wordt gehouden met het teken. Het resultaat is altijd groter of gelijk aan nul.

8.3. Argumenten

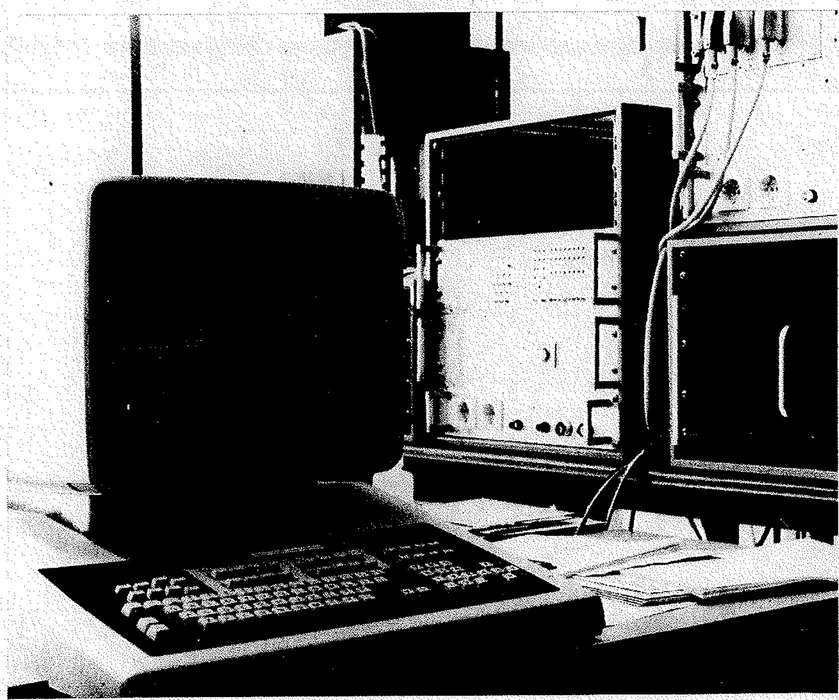
We zien nog andere kenmerken, die voor een functie algemeen zijn. De waarde waarop ABS(X) werkt, wordt tussen haken achter ABS weergegeven. Zo'n waarde heet een *argument.*
Een functie werkt dus op een argument. Zo'n argument kunnen we beschouwen als de invoerwaarde van de functie.

Afb. 8.1. Argument en resultaat van een functie



We zien ook dat de functie een resultaat aflevert (zie afbeelding 8.1). In bovenstaand voorbeeld wordt in regel 10 de waarde van het resul-

Microcomputersysteem voor algemene toepassingen



taat toegekend aan de variabele A. In regel 30 wordt het resultaat van de functie rechtstreeks uitgeprint op het scherm van de computer.

Een ander voorbeeld van het gebruik van ABS(X)

```
*LIST
10    REM DEMONSTRATIE VAN DE ABS FUNCTIE
20    PRINT "TYPE EEN GETAL"
30    INPUT K:PRINT
40    PRINT "ABSOLUTE WAARDE HIERVAN IS:";
50    PRINT ABS(K)
60    IF K<>(-1.0) GOTO 20
70    END
*RUN
TYPE EEN GETAL
?20
ABSOLUTE WAARDE HIERVAN IS: 20.0
TYPE EEN GETAL
?-20
ABSOLUTE WAARDE HIERVAN IS: 20.0
TYPE EEN GETAL
?0
ABSOLUTE WAARDE HIERVAN IS: 0.0
TYPE EEN GETAL
?-0
ABSOLUTE WAARDE HIERVAN IS: 0.0
TYPE EEN GETAL
?0.8
ABSOLUTE WAARDE HIERVAN IS: 0.8
TYPE EEN GETAL
?-1
ABSOLUTE WAARDE HIERVAN IS: 1.0
END PROGRAM
*
```

Hier zien we een voorbeeld van de voor de Teleac/DAI personal computer kenmerkende regel INPUT K: PRINT (regel 30). Vrijwel elke andere BASIC versie zal INPUT K gebruiken. We zien ook 20.0 uitgeprint als we 20 intypen. Dat is eveneens kenmerkend.
Vaak wordt ABS(X) gebruikt om een bepaalde test in minder statements te laten plaatsvinden. Wanneer we willen weten of een waarde W tussen −25 en +25 ligt, kunnen we eerst door middel van een IF statement testen of de waarde groter of gelijk is aan 25 en daarna of de waarde kleiner is dan −25.
We kunnen hetzelfde resultaat bereiken door het volgende statement:

```
IF ABS(W) < 25 GOTO ...
```

Tot nu toe hebben we alleen gezien dat het argument als constante of als variabele is weergegeven. BASIC accepteert als argument van een functie ook een expressie. Het is dus correct om tussen de haken een hele expressie te zetten zoals in onderstaand voorbeeld (regel 50):

```
*LIST
10    REM DEMONSTRATIE EXPRESSIE IN EEN ARGUMENT
20    PRINT "TYPE EEN GETAL"
30    INPUT K:PRINT
40    PRINT "DE ABSOLUTE WAARDE VAN HET GETAL MINUS 5 IS:";
50    PRINT ABS(K-5.0)
60    IF K<>(-1.0) GOTO 20
70    END
*RUN
TYPE EEN GETAL
?6
DE ABSOLUTE WAARDE VAN HET GETAL MINUS 5 IS: 1.0
TYPE EEN GETAL
?5
DE ABSOLUTE WAARDE VAN HET GETAL MINUS 5 IS: 0.0
TYPE EEN GETAL
?4
DE ABSOLUTE WAARDE VAN HET GETAL MINUS 5 IS: 1.0
TYPE EEN GETAL
?-2
DE ABSOLUTE WAARDE VAN HET GETAL MINUS 5 IS: 7.0
TYPE EEN GETAL
?-1
DE ABSOLUTE WAARDE VAN HET GETAL MINUS 5 IS: 6.0
END PROGRAM
*
```

Iedere in BASIC toegestane expressie kan als argument worden gebruikt.

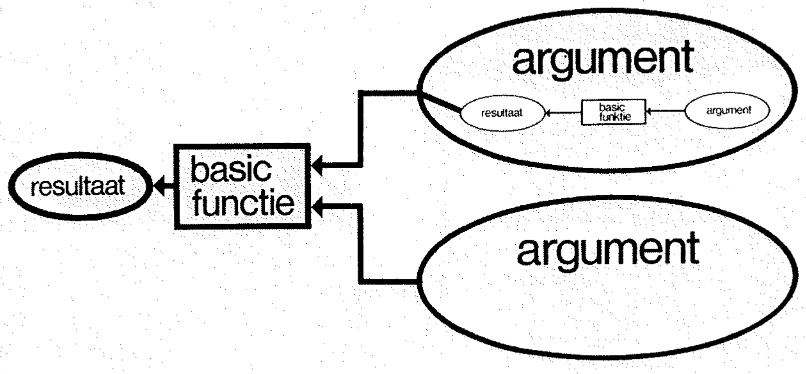
8.4. Resultaat

Een ander opmerkelijk feit van het gebruik van een functie is het volgende: Wanneer de functie is uitgevoerd, wordt hij als het ware vervangen door zijn resultaat. In bovenstaand voorbeeld zien we op regel 50 dat ABS (K−5) als het ware wordt vervangen door het resultaat van de berekening en dus ook geprint kan worden. Deze waarde kan weer als argument voor een functie dienen of aan een variabele worden toegekend. Dit betekent in feite dat we een functie deel uit kunnen laten maken van een expressie. Zo kunnen we ingewikkelde expressies vormen.
Wanneer we nu de opmerkingen over het gebruik van functies in BASIC samenvatten, dan zien we:

a. Een functie werkt op een argument. Het argument moet tussen haken staan. Het argument mag uit een expressie bestaan.
b. Het resultaat van een functie mag opnieuw in een expressie worden gebruikt.
c. Iedere functie werkt op één of meer argumenten en levert één resultaat af.
   In de meeste gevallen wordt bij onze computer op één argument gewerkt (zie afbeelding 8.2).

In dit hoofdstuk worden een aantal algemene BASIC functies behandeld. In de eerste plaats zijn dat de functies ABS(X), INT(X),

SGN(X), FRAC(X), SQR(X) en RND(X). Voor de meer mathematisch onderlegden, die bijvoorbeeld weten om te gaan met de zogenaamde wetenschappelijke pocketcalculator, gaan we in op de functies: SIN(X), COS(X), TAN(X), ASIN(X), ACOS(X), ATN(X), EXP(X), LOG(X) en LOGT(X).

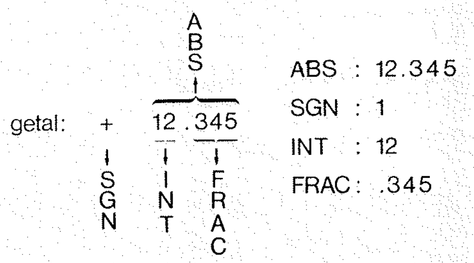


Afb. 8.2. Functies in een functie

In de *prioriteitenregels* wordt een functie net als haakjespaar beschouwd. De functies worden dus eerst uitgewerkt.

8.5. Eenvoudige functies

ABS(X)
De functie ABS(X) hebben we hierboven reeds behandeld; dat hoeft nu niet te worden herhaald. We zullen nu de functies, die op onderdelen van een getal werken, stuk voor stuk behandelen. Een overzicht van deze functies is in afbeelding 8.3 gegeven.



Afb. 8.3. Onderdelen van een getal

INT(X)
De waarschijnlijk meest gebruikte functie uit de eerstgenoemde groep is de functie INT(X). Het resultaat van INT(X) is de gehele waarde (in het Engels: integer) van het getal X.

Voorbeeld

```
*LIST
10     PRINT INT(10.23)
20     PRINT INT(10.0)
30     PRINT INT(-10.23)
*RUN
 10.0
 10.0
-11.0
*
```

We zien hieruit dat INT(X) als resultaat het grootste gehele getal geeft dat kleiner is dan het argument of dat daaraan gelijk is. Let dus vooral op bij negatieve getallen!
We zien dan uiteraard dat INT (−10.23) de waarde −11 als resultaat heeft, terwijl INT (10.23) de waarde +10 oplevert.
Indien we een getal X willen afronden tot D decimalen, kan dat door middel van de uitdrukking:

INT(X*10↑D+0.5)/10↑D

Voorbeeld

```
*LIST
10    REM AFRONDING M.B.V. DE INT FUNCTIE
20    PRINT "TYPE EEN GETAL IN"
30    INPUT X:PRINT
40    PRINT "DECIMALEN","AFRONDING"
50    FOR D=1.0 TO 5.0
60    LET R=INT(X*10.0↑D+0.5)/10.0↑D
70    PRINT D,R
80    REM NU DE VOLGENDE DECIMAAL
90    NEXT D
100   END
*RUN
TYPE EEN GETAL IN
?1.23456
DECIMALEN    AFRONDING
 1.0          1.2
 2.0          1.23
 3.0          1.235
 4.0          1.2346
 5.0          1.23456
END PROGRAM
*
```

Als we willen nagaan of een getal deelbaar is door een ander getal, kan dat ook met de functie INT(X).
Indien we willen weten of X een geheel veelvoud is van Y, gebruiken we:

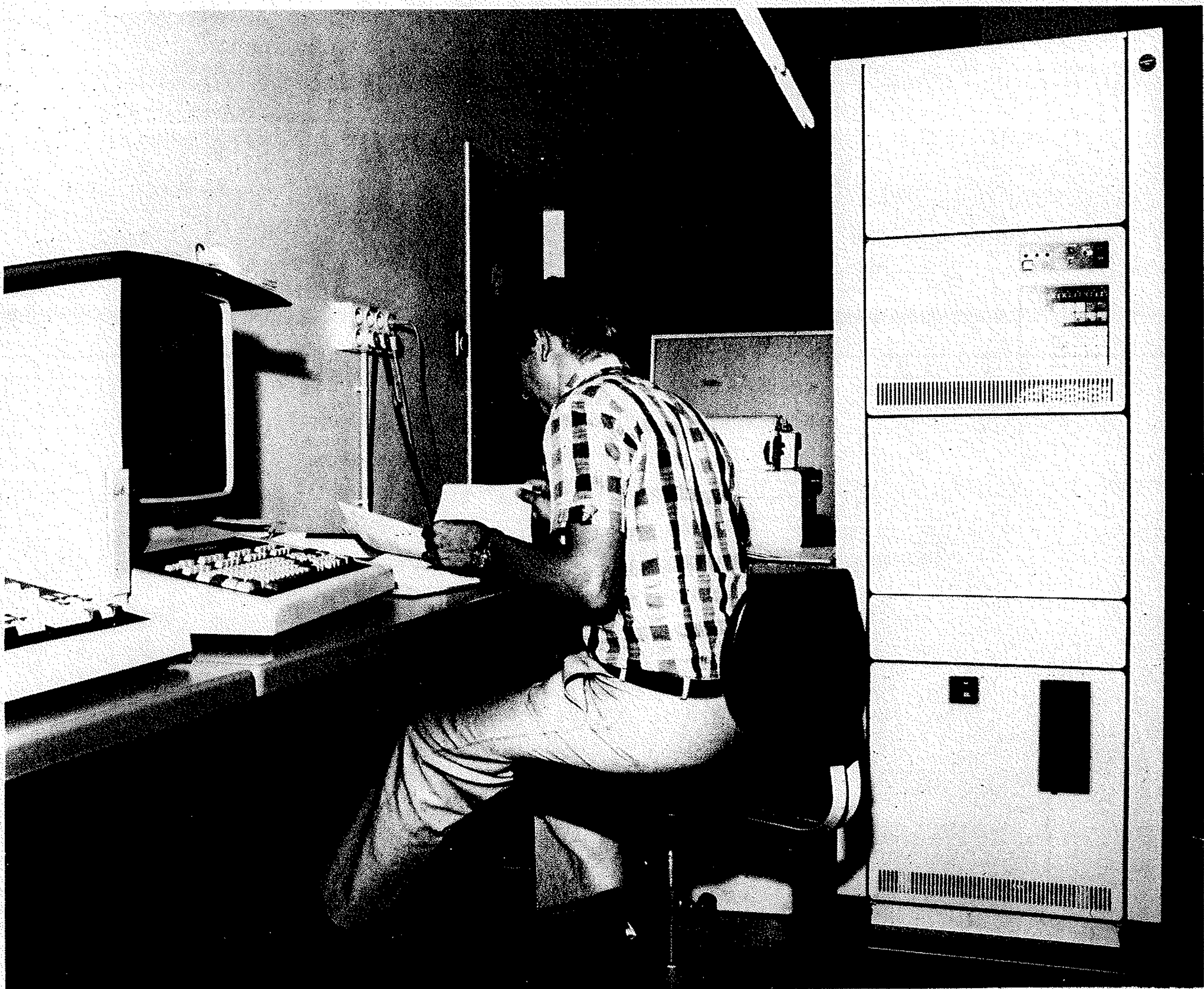
```
*LIST
10    PRINT "TYPE EEN GETAL"
20    INPUT X:PRINT
30    PRINT "NOG EEN GETAL"
40    INPUT Y:PRINT
50    IF (X/Y)=INT(X/Y) GOTO 70
55    PRINT
60    GOTO 80
70    PRINT X;" IS DEELBAAR DOOR";Y
80    IF X<>(-1.0) THEN 10
90    END
*RUN
TYPE EEN GETAL
?5
NOG EEN GETAL
?7
TYPE EEN GETAL
?10
NOG EEN GETAL
?5
 10.0 IS DEELBAAR DOOR 5.0
TYPE EEN GETAL
?-1
NOG EEN GETAL
?0
DIVISION BY ZERO IN LINE 50
*
```

SGN(X)
De volgende functie in deze lijst is de functie SGN *(sign = signum = teken)*. Deze functie geeft het teken van het argument weer. Dus voor SGN(X) geldt als resultaat 1 indien X > 0; 0 indien X = 0; − 1 indien X < 0, zoals wordt gedemonstreerd door:

```
*LIST
10    PRINT SGN(10.23)
20    PRINT SGN(0.0)
30    PRINT SGN(-10.23)
*RUN
 1.0
 0.0
-1.0
*
```

We hebben nu gezien hoe we de gehele waarde van een getal kunnen krijgen. Daarna is getoond hoe we het teken ervan kunnen verwerven.

Een IBM-minicomputer die meerdere microcomputers bevat.



etal,

ken

FRAC(X)
De *fractie* is dat deel van een getal dat we achter de komma (punt) aantreffen. Daartoe dient de functie FRAC(X).

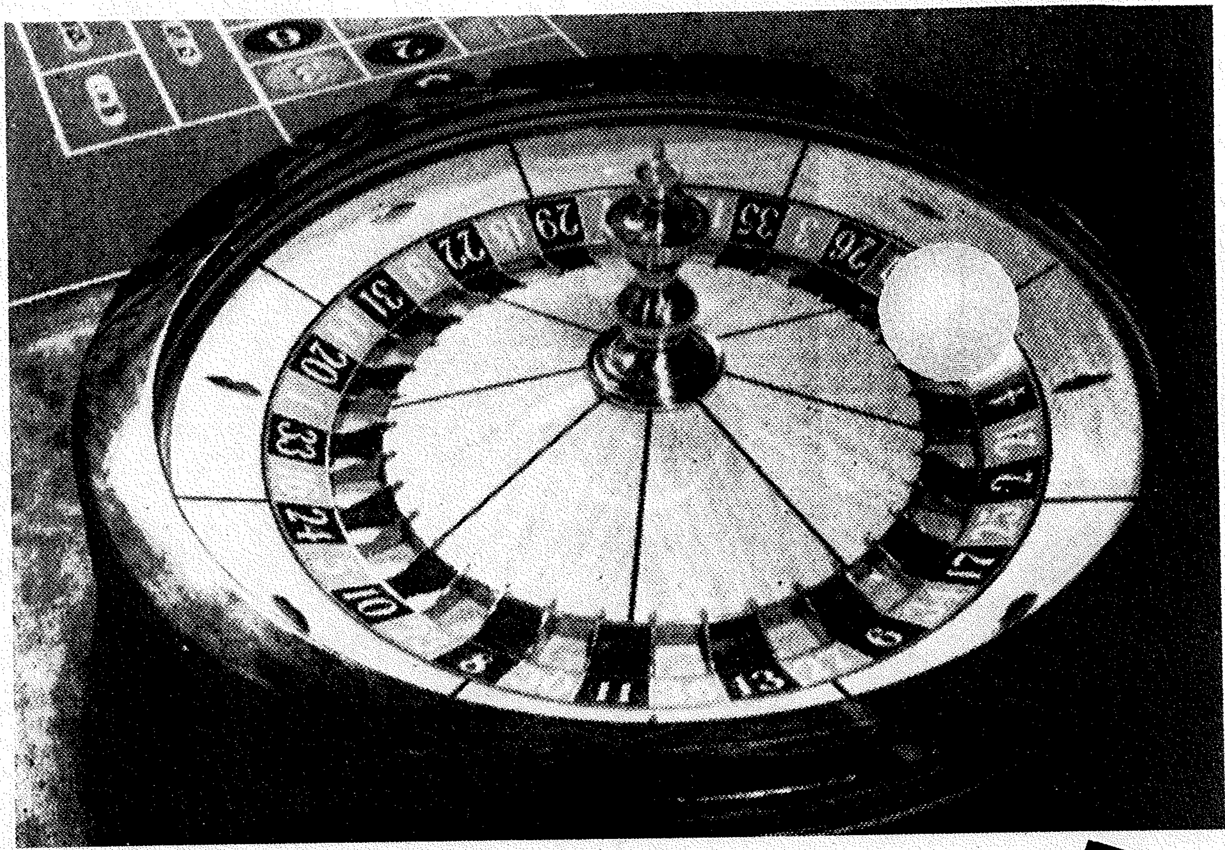
Voorbeeld

```
*LIST
10      PRINT FRAC(10.23)
20      PRINT FRAC(-10.23)
30      PRINT FRAC(+10.0)
40      PRINT FRAC(-10.0)
*RUN
 0.23
-0.23
 0.0
 0.0
*
```

Niet alle personal computers kennen de functie FRAC. Raadpleeg weer de handleiding. Let wel dat FRAC met gebruik van de functie INT ook kan worden berekend. Oefent u dat maar eens en let vooral op bij negatieve getallen.

SQR(X)
De volgende functie is de functie *vierkantsworteltrekken* (in het Engels *square root).* Deze wordt weergegeven door SQR.
Voor Y = SQR(X) geldt dat Y = √X ofwel Y * Y = X.
Het argument mag hier dus niet negatief zijn. Het volgende programma geeft de resultaten van SQR(X) weer:

```
*LIST
10      REM DEMONSTRATIE VAN SQR
20      FOR X=1.0 TO 10.0
30      R=SQR(X)
40      PRINT X,R,R*R
50      NEXT X
60      END
*RUN
 1.0          1.0          1.0
 2.0          1.41421      2.0
 3.0          1.73205      3.0
 4.0          2.0          4.0
 5.0          2.23607      5.0
 6.0          2.44949      6.0
 7.0          2.64575      7.0
 8.0          2.82843      8.0
 9.0          3.0          9.0
 10.0         3.16228      10.0
END PROGRAM
*
```



RND

Afb. 8.4. Random generator

RND(X)
Een belangrijke functie is RND(X), die als het ware in staat is willekeurige getallen te geven. (RND = *random* = *willekeurig*).
RND (X) genereert een getal tussen 0 en X in de Teleac/DAI personal computer.

Voorbeeld

```
*LIST
10    PRINT RND(3.0)
20    PRINT RND(3.0)
30    PRINT RND(5.0)
*RUN
 2.56537
 0.514494
 0.291583
*RUN
 1.96288
 1.4994
 2.08744
*
```

De getallen, die geproduceerd worden, zijn zogenaamde *pseudo random getallen*. Een random getal is onvoorspelbaar. Aangezien de getallen, die door de RND(X) functie worden geproduceerd, door een vast geprogrammeerd programma worden geleverd, zijn ze niet echt random, maar pseudo random. Statistisch gezien lijken ze ech-

ter sterk op echte random getallen. We kunnen op deze manier een onvoorspelbaar gedrag van een programma introduceren (zie afbeelding 8.4).
We kiezen een voorbeeldprogramma, dat zich gedraagt als het trekken van een lootje uit een hoed. We veronderstellen dat er 10 lootjes, genummerd van 1 tot 10, aanwezig zijn en dat we vijfmaal trekken. Na het trekken van het lootje leggen we het weer terug in de hoed.

```
*LIST
10    REM LOOTJES TREKKEN
20    REM 5 UIT 10 MET TERUGLEGGEN
30    FOR X=1.0 TO 5.0
40    LET T=INT(1.0+RND(10.0))
50    PRINT T,
60    NEXT X
*RUN
 2.0        10.0        2.0        4.0        6.0
*RUN
 5.0        10.0        1.0        2.0        1.0
*RUN
 6.0        4.0         10.0       9.0        5.0
*RUN
 3.0        4.0         4.0        4.0        4.0
*
```

In regel 40 tellen we eerst 1 bij het getal op, zodat de waarden tussen 1 en 11 komen te liggen. Vervolgens maken we er gehele getallen van opdat alleen de waarden 1 tot en met 10 kunnen ontstaan.
Er zijn tussen personal computers onderling verschillen in de mogelijkheden van BASIC en ook in de definities van de BASIC statements, functies, enz.
De Teleac/DAI personal computer wijkt bij RND(X) af van de meest voorkomende definitie. In de meeste personal computers zal RND(X) een getal tussen 0 en 1 opleveren. Als X daar positief is, wordt bij ieder RND(X) executie een nieuwe randomwaarde tussen 0 en 1 geproduceerd. Bij X negatief wordt de vorige randomwaarde herhaald. Bij X = 0 levert RND(X) steeds dezelfde waarde op. Dit laatste kan bij het testen van een programma prettig zijn. Er is dan met zekerheid bekend welke waarde RND (0) levert.
Veel spelletjes die in BASIC geprogrammeerd kunnen worden, zijn gebaseerd op het gebruik van de RND (X) functie.
Het kiezen uit de getallen 0 t/m 5 kunnen we nabootsen met INT (RND(6)):

```
*LIST
90    FOR X=1.0 TO 10.0
100   PRINT INT(RND(6.0));" ";
110   NEXT X
*RUN
 1.0  0.0  1.0  1.0  5.0  5.0  3.0  0.0  1.0  5.0
*RUN
 2.0  4.0  0.0  0.0  0.0  1.0  2.0  3.0  3.0  1.0
*
```

Een dobbelsteen kan dus worden nagebootst met 1 + INT(RND(6)).
Als we getallen van A tot B laten genereren, is een bijbehorend programma:

```
*LIST
10    PRINT "KLEINSTE GETAL"
20    INPUT A:PRINT
30    PRINT "GROOTSTE GETAL"
40    INPUT B:PRINT
50    FOR X=1.0 TO 10.0
60    PRINT A+INT(RND(B-A));" ";
70    NEXT X
*RUN
KLEINSTE GETAL
?4
GROOTSTE GETAL
?24
 11.0  21.0  12.0  20.0  15.0  11.0  19.0  20.0  8.0  14.0

*RUN
KLEINSTE GETAL
?2
GROOTSTE GETAL
?78
 58.0  35.0  55.0  11.0  4.0  41.0  5.0  53.0  15.0  28.0
*
```

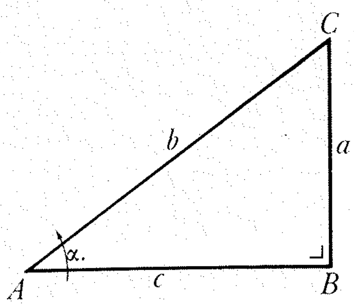
Afbeelding 8.5 geeft de nu behandelde algemene functies nog eens in een tabelvorm weer.

ABS (X)	**geeft de absolute waarde van X**
INT(X)	**geeft de grootste gehele waarde kleiner of gelijk aan X**
SGN(X)	**geeft het teken van X, of geeft aan dat X gelijk aan Ø is**
FRAC(X)	**geeft de fractie van X**
SQR(X)	**geeft de wortel van X**
RND(X)	**geeft een pseudo random getal**
FRE	**geeft de hoeveelheid vrij geheugen**

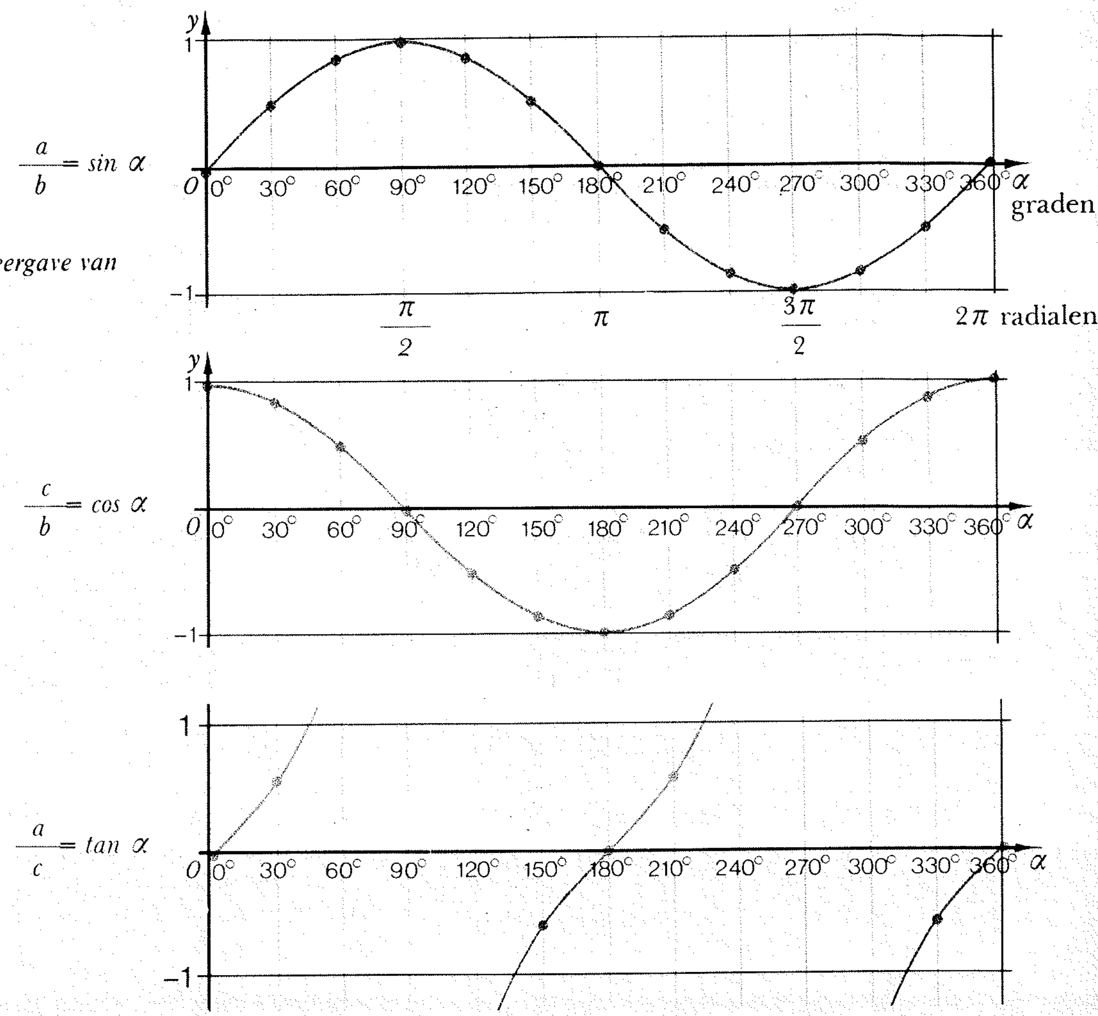
Afb. 8.5. Algemene functies

8.6. Goniometrische functies

De tweede groep functies, die we behandelen, zijn de zogenaamde *goniometrische* functies. Voor hen die in deze materie niet thuis zijn, wordt verwezen naar de leerboeken van het middelbaar onderwijs, het technisch onderwijs, enz.
Als geheugensteuntje geven we toch nog even de betekenis van verschillende hier te gebruiken functies. Bekijk de rechthoekige driehoek zoals is weergegeven in afbeelding 8.6, vooral hoek α.
Het verband tussen de hoek α en de sinus, cosinus en tangens van α over α lopend van 0 tot 360 graden (0 tot 2π radialen) is weergegeven in afbeelding 8.7.



Afb. 8.6. Driehoek voor goniometrische functies



Afb. 8.7. Grafische weergave van sin α, cos α en tan α

SIN(X)

De eerste functie uit deze rij is de *sinus*. Deze wordt weergegeven door SIN(X). X kan weer een expressie zijn. X wordt geïnterpreteerd als radialen. In het volgende voorbeeldprogramma geven we de sinus van 0 tot 20 graden op ons beeldscherm weer.

```
*LIST
10    REM SINUSDEMONSTRATIE
20    REM WE DRUKKEN NU PER GRAAD AF
30    PRINT "GRADEN","SINUS"
40    FOR I=0.0 TO 20.0
50    PRINT I,SIN(I/57.2958)
60    NEXT I
70    END
*
*RUN
```

(vervolg programma)

```
GRADEN        SINUS
 0.0           0.0
 1.0           1.74524E-2
 2.0           3.48995E-2
 3.0           5.23359E-2
 4.0           6.97565E-2
 5.0           8.71557E-2
 6.0           0.104528
 7.0           0.121869
 8.0           0.139173
 9.0           0.156434
 10.0          0.173648
 11.0          0.190809
 12.0          0.207912
 13.0          0.224951
 14.0          0.241922
 15.0          0.258819
 16.0          0.275637
 17.0          0.292372
 18.0          0.309017
 19.0          0.325568
 20.0          0.34202
END PROGRAM
*
```

Op regel 50 zien we dat er een omrekening van graden naar radialen plaatsvindt. 1 radiaal = $180/\pi = 57.2958$ graden ofwel 180 graden = π radialen. De sinus van X graden is dus SIN (X/57.2958).

COS(X)
Verder beschikken we over de *cosinus*, waarbij het argument opnieuw in *radialen* moet worden uitgedrukt.

Voorbeeld

```
*LIST
10    REM VOORBEELD COSINUS
20    PRINT "TYP EEN WAARDE"
30    INPUT G:PRINT
40    PRINT "DE COSINUS VAN";G;" IS";COS(G)
50    IF G<>0.0 GOTO 20
*RUN
TYP EEN WAARDE
?89
DE COSINUS VAN 89.0 IS 0.510181
TYP EEN WAARDE
?45
DE COSINUS VAN 45.0 IS 0.525321
TYP EEN WAARDE
?0
DE COSINUS VAN 0.0 IS 1.0
*
```

In dit voorbeeld vraagt het programma steeds opnieuw een waarde, totdat 0 wordt ingetypt.

TAN(X)
Tenslotte beschikken we uiteraard ook over de *tangens*. Deze wordt weergegeven door TAN(X), waarbij X ook weer in radialen is uitgedrukt.

Voorbeeld
```
*LIST
10    PRINT TAN(10.0/57.2958)
20    PRINT TAN(0.0)
30    PRINT TAN(2.0)
*RUN
 0.176327
 0.0
-2.18504
*
```

PI
In het voorbeeld van de sinus zagen we dat de vertaling van radialen naar graden via de formule $180/\pi$ verloopt.
Het getal pi (π) zelf is ook in het systeem aanwezig onder de naam PI. De waarde daarvan is 3.14159... zoals we zien in onderstaand voorbeeld:

```
*LIST
10    PRINT PI
*RUN
 3.14159
*
```

In plaats van het getal 57.2985 kunnen we bij de omrekening ook gewoon gebruik maken van de expressie 180/PI.

ASIN(X)
Vervolgens beschikken we over de functie *boogsinus* (in het Engels *arcsine)* ofwel ASIN(X).
Het resultaat in deze functie ligt in BASIC altijd tussen $-\pi/2$ en $+\pi/2$. (zie voorbeeld op blz. 141)

De gebruiker moet er zich van bewust zijn dat een computer uiteraard een beperkte rekennauwkeurigheid bezit. Het aantal cijfers achter de komma waarmee wordt gerekend, is namelijk beperkt. De uitgeprinte eerste en derde waarden op een regel zouden alle gelijk moeten zijn. Door afrondingsfouten in het systeem kunnen echter afwijkingen ontstaan, die zich in dit voorbeeld evenwel niet manifesteren.
In de mate van afrondingsfouten kunnen tussen personal computers duidelijke verschillen worden gevonden. De verschillen komen voort uit de manier, waarop intern in het systeem wordt gerekend. De Teleac/DAI personal computer werkt voor gebroken getallen met een nauwkeurigheid van 6 cijfers, zoals de meeste andere personal computers. Het 6e cijfer is dus steeds afgerond.

Voorbeeld

```
*LIST
10    REM BOOGSINUS VOORBEELD
20    PRINT "W","ASIN(W)","SIN(ASIN(W))"
30    FOR W=-0.5 TO 0.5 STEP 0.1
40    PRINT W,ASIN(W),SIN(ASIN(W))
50    NEXT W
60    END
*RUN
W              ASIN(W)        SIN(ASIN(W))
-0.5           -0.523599      -0.5
-0.4           -0.411517      -0.4
-0.3           -0.304693      -0.3
-0.2           -0.201358      -0.2
-0.1           -0.100167      -0.1
-1.49012E-8    -1.49012E-8    -1.49012E-8
 0.1            0.100167       0.1
 0.2            0.201358       0.2
 0.3            0.304693       0.3
 0.4            0.411517       0.4
 0.5            0.523599       0.5
END PROGRAM
*
```

Voor *integer getallen* werkt de Teleac/DAI personal computer evenwel met 9 cijfers. Dit is gedaan om nauwkeurige berekeningen, bijvoorbeeld geldswaarden, toe te staan *(integer = geheel getal)*.
Een variabele kan als integer aangeduid worden door het afsluiten van de naam met een % teken, dus bij voorbeeld A%, B%, enz.

```
*LIST
10    INPUT "A";A:PRINT
20    PRINT A,A+A
30    A%=A
40    PRINT A%,A%+A%
*RUN
A?1234
 1234.0        2468.0
 1234          2468
*RUN
A?123456789
 1.23457E8     2.46914E8
 123456792     246913584
*
```

De resultaten van berekeningen met integer variabelen worden weer als integer weergegeven, zoals uit de 2e run blijkt. Bij A wordt de wetenschappelijke notatie gebruikt, bij A% niet. In de handleiding van de computer staat precies aangegeven, welke functies een integer resultaat kunnen produceren. In het algemeen zijn dat de 'gewone' rekenkundige functies en operaties. De meer complexe functies en operaties, zoals de sinus, enz., produceren nooit een integer resultaat, maar een *floating point* (ofwel gebroken) getal.

ACOS(X)
Behalve de boogsinus beschikken we ook over de *boogcosinus:* ACOS (X). Het resultaat hiervan ligt in BASIC tussen 0 en PI.

Voorbeeld

```
*LIST
10      PRINT ACOS(0.0)
*RUN
 1.5708
*
```

ATAN(X)
Tenslotte beschikken we in principe ook over de *boogtangens*-functie ATAN(X). Deze levert een resultaat tussen −PI/2 en +PI/2. Wanneer uw computer deze functie niet kent, dan kunt u hem berekenen volgens de formule:

ATAN (W) = ASIN (W/SQR(W * W + 1))

Voor de verandering laten we de ATAN(X) maar eens op deze wijze berekenen, hoewel de Teleac/DAI personal computer over de functie ATAN(X) beschikt.

Voorbeeld

```
*LIST
10      REM BOOGTANGENS VOORBEELD
15      REM ATAN(W)=ASIN(W/SQR(W*W+1))
20      PRINT "WAARDE","ATAN"
30      FOR W=0.0 TO 100.0 STEP 10.0
40      PRINT W,ASIN(W/SQR(W*W+1.0))
50      NEXT W
60      END
*RUN
WAARDE        ATAN
 0.0           0.0
 10.0          1.47113
 20.0          1.52084
 30.0          1.53747
 40.0          1.5458
 50.0          1.5508
 60.0          1.55413
 70.0          1.55651
 80.0          1.5583
 90.0          1.55969
 100.0         1.56079
END PROGRAM
*
```

EXP(X)
In deze categorie blijven nog de functies *exponent* en *logarithme* over. De functie EXP(X) geeft de constante e (=2.71828) tot de macht X weer (e^x).

Voorbeeld

```
*LIST
10    PRINT EXP(0.0),EXP(1.0),EXP(1.23),EXP(-1.23)
*RUN
 1.0          2.71828      3.42123      0.292293
*
```

LOG(X), LOGT(X)

De eerste functie is de *natuurlijke logarithme* ln X(basis e): LOG(X). Dat wil zeggen dat voor y = LOG(X) geldt: X = e^y.

Voorbeeld

```
*LIST
10    PRINT LOG(EXP(1.0))
20    PRINT LOG(10.0)
30    PRINT LOG(0.0)
*RUN
 1.0
 2.30259
NUMBER OUT OF RANGE IN LINE 30
*
```

De logarithme van 0 bestaat niet, hetgeen ook door de computer wordt gesignaleerd.
Wanneer we de basis Y logarithme van X wensen, kan dat via de formule LOG(X)/LOG(Y). Als voorbeeld geven we de gewone 10 logarithme van 3:

LOG(3)/LOG(10)

Omdat de 10 logarithme veel wordt gebruikt, is deze ook in de Teleac/DAI personal computer onder de naam LOGT aanwezig.

Voorbeeld

```
*LIST
10    PRINT LOGT(10.0)
20    PRINT LOGT(100.0)
*RUN
 1.0
 2.0
*
```

Bovengenoemde wetenschappelijke functies worden in afbeelding 8.8 nog eens samengevat. Met behulp van de behandelde functies kunnen we ook zelf nog andere gebruikelijke functies berekenen.

SIN(X)	geeft de sinus van X
COS(X)	geeft de cosinus van X
TAN(X)	geeft de tangens van X
ASIN(X)	geeft de boogsinus van X
ACOS(X)	geeft de boogcosinus van X
ATAN(X)	geeft de boogtangens van X
EXP(X)	geeft de e tot de macht X (e^X)
LOG(X)	geeft de natuurlijke logarithme van X (ln X)
LOGT(X)	geeft de 10 - logarithme van X

Afb. 8.8. Wetenschappelijke functies

Wellicht ten overvloede vermelden we een aantal daarvan in afbeelding 8.9.

```
ARCSEC(X) = ATN(SQR(X*X - 1)) + (SGN(X) - 1)*1.5708
ARCCSC(X) = ATN(1/SQR(X*X) - 1)) + (SGN(X) - 1)*1.5708
ARCCOT(X) = -ATN(X) + 1.5708
ARCSINH(X) = LOG(X + SQR(X*X + 1))
ARCCOSH(X) = LOG(X + SQR(X*X - 1))
ARCTANH(X) = LOG((1 + X)/(1 - X))/2
ARCSECH(X) = LOG((SQR(X*X + 1) + 1)/X)
ARCCSCH(X) = LOG((SGN(X)*SQR(X*X + 1) + 1)/X)
ARCCOTH(X) = LOG((X + 1/(X - 1))/2
COT(X) = 1/TAN(X)
CSC(X) = 1/SIN(X)
SEC(X) = 1/COS(X)
COSH(X) = (EXP(X) + EXP( - X))/2
COTH(X) = EXP(-X)/(EXP(X) - EXP( - X))*2 + 1
CSCH(X) = 2/(EXP(X) - EXP(-X))
SECH(X) = 2/(EXP(X)+ EXP(-X))
SINH(X) = (EXP(X) - EXP(-X))/2
TANH(X) = -EXP(-X)/(EXP(X) + EXP(-X))*2 + 1
```

Afb. 8.9. Afgeleide wetenschappelijke functies

8.7. Algemene opmerkingen

In verschillende personal computers bestaat de mogelijkheid om zelf in een programma een functie te definiëren. Daarbij wordt dan het statement DEF (ofwel define) gebruikt. Met behulp van dit statement worden de naam van de zelf gedefinieerde functie en de argumenten, die erbij horen, vastgelegd.

Voorbeeld

DEF COT (X) = 1/TAN(X)

Hierbij wordt een functie COT (cotangens) gedefinieerd. Het argument van deze functie is X. Het resultaat wordt berekend als 1/TAN(X).

Wanneer we in een programma deze functie COT(X) zo gedefinieerd hebben, kan hij net als de reeds aanwezige functies worden gebruikt.

Voorbeeld

50 PRINT COT (0.123) + 0.5

De te printen waarde zal hier zijn (1/TAN(0.123)) + 0.5

Soms is het zelfs mogelijk een functie te definiëren die uit meer regels bestaat. Deze wordt dan afgesloten met FNEND. Raadpleeg steeds de handeling van uw computer.

De Teleac/DAI personal computer kent deze mogelijkheid evenwel niet. De functies DEF en FNEND zijn daar onbekend en zullen aanleiding tot een foutmelding geven.

Samenvatting hoofdstuk 8

In dit hoofdstuk zijn een groot aantal BASIC functies behandeld. We begonnen met functies die getallen uiteenrafelen. Daarna werd de randomfunctie behandeld. Tenslotte hebben we de wetenschappelijke functies bekeken en gezien hoe we andere goniometrische functies uit de hier gegeven functies kunnen berekenen.

Multiple choice-vragen bij hoofdstuk 8

A. De 'BASIC-functie' wordt ook wel met andere namen aangeduid. Welke naam hoort daarbij *niet* in onderstaand rijtje thuis?

1 intrinsic function
2 library function
3 subroutine
4 bibliotheekfunctie

B. Hier worden kenmerken van een functie genoemd. Welk kenmerk hoort in onderstaand rijtje *niet* thuis?

1 terugkeer via RETURN
2 gebruik van een of meer argumenten
3 een in expressies bruikbaar resultaat
4 aanroep onder naam

C. a. Een expressie mag als argument dienen voor een functie.
b. Een functie mag worden gebruikt als argument voor een functie.

1 uitspraak a is goed en uitspraak b is goed
2 uitspraak a is fout en uitspraak b is goed
3 uitspraak a is goed en uitspraak b is fout
4 uitspraak a is fout en uitspraak b is fout

D. a. ABS (X), INT (X), SGN (X) en FRAC (X) werken elk op een afzonderlijk onderdeel van een getal X.
b. SGN (X) geeft alleen aan of een getal positief of negatief is.

1 uitspraak a is goed en uitspraak b is goed
2 uitspraak a is fout en uitspraak b is goed
3 uitspraak a is goed en uitspraak b is fout
4 uitspraak a is fout en uitspraak b is fout

E. a. SGN (X) * INT (ABS (X)) is gelijkwaardig aan INT (X).
b. FRAC (X) is gelijkwaardig aan X−INT (X).

1 uitspraak a is goed en uitspraak b is goed
2 uitspraak a is fout en uitspraak b is goed
3 uitspraak a is goed en uitspraak b is fout
4 uitspraak a is fout en uitspraak b is fout

F. **a.** RND genereert pseudo-random getallen.
b. RND(X) genereert gehele getallen tussen 0 en X in de Teleac/DAI-personal computer.

1 uitspraak a is goed en uitspraak b is goed
2 uitspraak a is fout en uitspraak b is goed
3 uitspraak a is goed en uitspraak b is fout
4 uitspraak a is fout en uitspraak b is fout

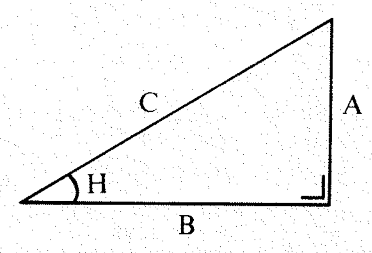
G. **a.** ABS(X) geeft altijd een positief resultaat.
b. PI heeft geen argument nodig.

1 uitspraak a is goed en uitspraak b is goed
2 uitspraak a is fout en uitspraak b is goed
3 uitspraak a is goed en uitspraak b is fout
4 uitspraak a is fout en uitspraak b is fout

H. **a.** Voor Y = LOGT (X) geldt X = 10 ↑ Y.
b. Voor Y = LOGT (X) geldt X = EXP (Y).

1 uitspraak a is goed en uitspraak b is goed
2 uitspraak a is fout en uitspraak b is goed
3 uitspraak a is goed en uitspraak b is fout
4 uitspraak a is fout en uitspraak b is fout

I.



a. COS (H) is in BASIC gelijkwaardig aan COS (ACOS(B/C)).
b. LOG (EXP (X)) is in BASIC gelijkwaardig aan X.

1 uitspraak a is goed en uitspraak b is goed
2 uitspraak a is fout en uitspraak b is goed
3 uitspraak a is goed en uitspraak b is fout
4 uitspraak a is fout en uitspraak b is fout

J. **a.** ABS(X), INT(X) en FRAC(X) kunnen allemaal het getal 0 als resultaat geven.
b. SQR (X), COS (X) en RND (X) kunnen allemaal het getal 0 als resultaat geven.

1 uitspraak a is goed en uitspraak b is goed
2 uitspraak a is fout en uitspraak b is goed
3 uitspraak a is goed en uitspraak b is fout
4 uitspraak a is fout en uitspraak b is fout

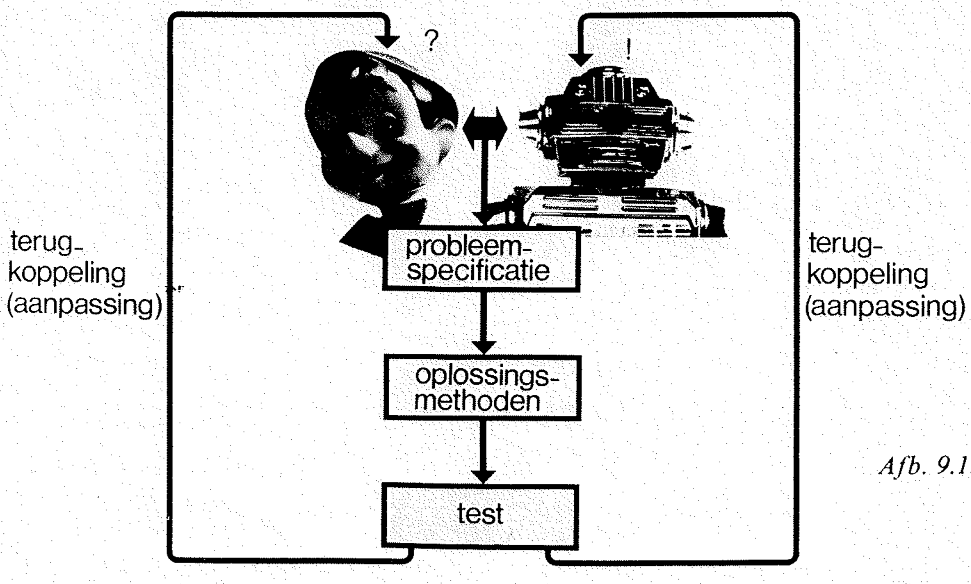
Hoofdstuk 9

Programma-opbouw

Programmeren in een taal is meer dan alleen maar enige instructies in die taal neerschrijven en dan zien of ze het beoogde resultaat opleveren. Het is in de eerste plaats belangrijk te beseffen dat het te maken programma een onderdeel vormt van een oplossing van een probleem. Het oplossen van dat probleem vergt meer stappen, waarvan we nu de belangrijkste zullen bekijken.

9.1. Probleemspecificatie

De eerste stap, die moet worden gemaakt bij het oplossen van een probleem, is het probleem goed te beschrijven. Het probleem moet daarbij *zo duidelijk mogelijk* worden beschreven. Dit vereist eveneens dat speciale te stellen eisen alle voorwaarden waaraan moet worden voldaan, helder geformuleerd zijn. We komen zo tot de specificaties van het op te lossen probleem. Het belang van de specificatie mag zeker niet worden onderschat.
Hoe dat probleem moet worden opgelost en waarmee, is nog niet aan de orde gekomen. Veel oplossingen in de praktijk blijken het probleem, dat de gebruiker voor ogen stond, niet echt op te lossen. In zeer veel gevallen is niet een verkeerde programmering van de computer of het falen van de computer zelf de oorzaak, maar de bron is een *onvolledige* probleemspecificatie.

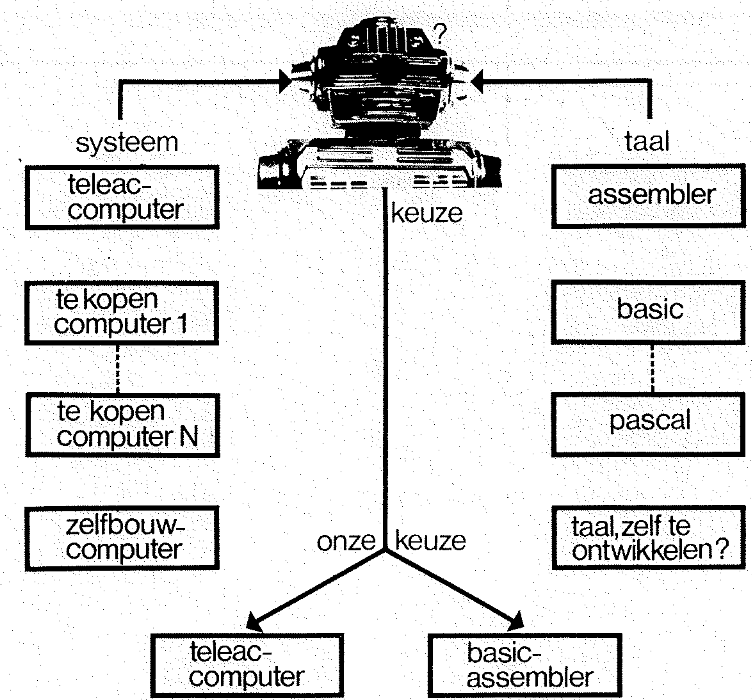


Afb. 9.1. Probleemdefinitie

Wanneer dan ook tijdens het werken aan de oplossingsmethode nog vragen naar voren komen ten aanzien van de specificatie, moeten deze eerst beantwoord worden, voordat men verder gaat. In de praktijk ligt ook hier dikwijls een oorzaak van ontevredenheid bij de uiteindelijke gebruiker. Degene, die voor het probleem een oplossing zoekt, kiest dikwijls op eigen initiatief een *aanvulling* van de specificatie, die hem of haar op dat moment het beste lijkt. Later blijkt dan dat dit niet overeenkomt met de wens van de gebruiker (zie afbeelding 9.1).
Het is niet de bedoeling hier een lijst met verordeningen te geven, maar om de lezer erop te wijzen dat een goede specificatie van het probleem onontbeerlijk is.
Vooral een beginnend programmeur heeft nogal eens de neiging om het schrijven van *statements* als het grote hoofdprobleem te zien. Zo'n programmeur zal echter, zeker na enige ervaring, beseffen dat het bijna altijd wel mogelijk is een oplossing, die voldoende concreet is geformuleerd, in statements om te zetten. Het benadrukken van het belang van een goede specificatie is zeer opzettelijk gedaan.
De specificatie zal met alle betrokkenen grondig moeten worden doorgepraat.

9.2. Hoe wordt het probleem opgelost?

Nadat de probleembeschrijving is afgehandeld, zal men een methode moeten genereren die het probleem volledig tot een goed einde voert. Met andere woorden: hoe ga ik het probleem oplossen? Daarvoor zijn gewoonlijk veel keuzemogelijkheden. In het kader van dit boek kunnen we daarbij van het volgende feit uitgaan:
– We moeten bij het oplossen van het probleem de Teleac/DAI personal computer gebruiken.



Afb. 9.2. Systeem- en taalkeuze

Dit geeft al een behoorlijke beperking. We hoeven namelijk niet meer te kiezen of we het probleem met een computer willen oplossen of niet. Als de oplossing door middel van een computer kan gebeuren, hoeven we ons niet meer te verdiepen in de keuze van een computer. Dat kan een omvangrijk probleem vormen, waarbij aspecten als prestatie, prijs, betrouwbaarheid en bruikbaarheid in de toekomst een grote rol spelen. We vragen ons hier dan ook niet af of we een computer moeten kopen of zelf moeten bouwen. In de meeste gevallen zal dit laatste niet aan de orde komen. Toch is het nuttig ook dit aspect in de praktijk te overwegen, met name wanneer het gaat om een apparaat, dat uiteindelijk in massaproduktie wordt gefabriceerd. In ons vereenvoudigd geval gaan we er dus vanuit dat we de Teleac/DAI personal computer gebruiken.
De meeste computers hebben een veelheid van programmeringsmogelijkheden, dat wil zeggen dat we kunnen kiezen uit een aantal *programmeertalen,* waarmee we ons probleem kunnen oplossen. Ook dit keuze-aspect zullen we in het kader van deze cursus negeren. We programmeren de Teleac/DAI personal computer met behulp van BASIC (zie afbeelding 9.2).
Doordat we deze keuze hebben gemaakt, is in feite ook vastgelegd hoe we onze probleemoplossing zowel in *hardware* als in *software* verdelen.
We hebben hier de specificatie en opbouw van een programma in enge zin bezien. Wanneer we in ruimere zin naar automatisering kijken, zien we vele fasen tijdens de oplossing van een probleem. Gewoonlijk worden dan de volgende stappen onderscheiden: vooronderzoek, analyse, ontwerp, ontwikkeling, invoering en evaluatie. Het *vooronderzoek* (ofwel toepasbaarheidsonderzoek) behelst de vraag: kan het probleem met een computer worden opgelost? *Analyse* kijkt naar alles, dat eisen stelt aan het te ontwerpen systeem. De *ontwerpfase* is de fase, waarin de opzet van de oplossing wordt gegeven. Een uitloper daarvan is de *ontwikkeling,* waarin in feite het ontwerp verder wordt uitgewerkt. Bij de *invoering* wordt de gebruiker geconfronteerd met de aangeboden oplossing. Tenslotte dient de *evaluatie* om na te gaan of de gebruiker tevreden is met de geboden oplossing of dat nog verbeteringen moeten worden aangebracht.
In feite moeten deze fasen altijd worden doorlopen. In het kader van dit boek zullen we ons concentreren op het ontwerp en de ontwikkeling.
In elk van deze fasen moeten gedachten en/of programma's uitgebreid worden getest. Het is dus beter het testen niet als een aparte fase te zien, maar als onderdeel van elke fase.

9.3. Software

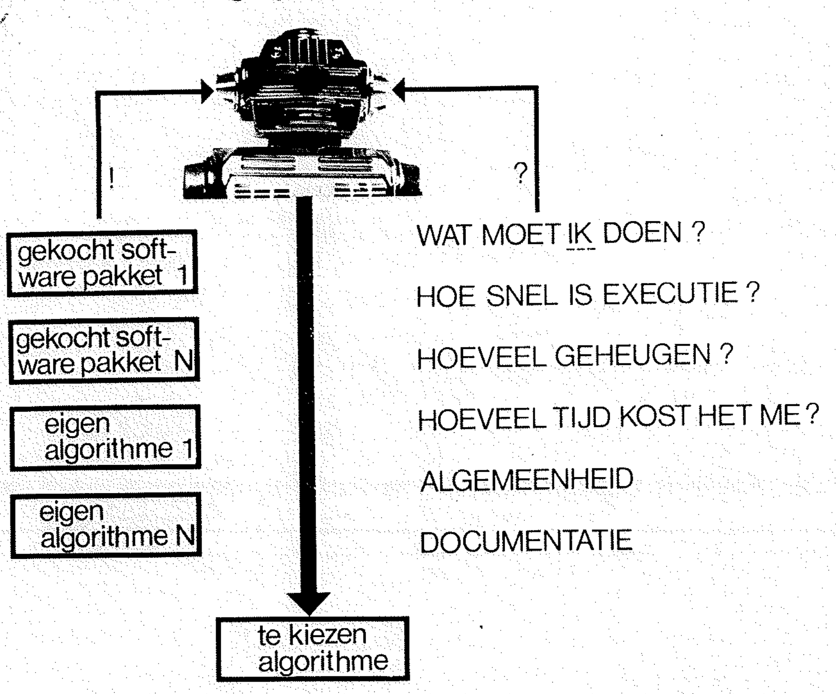
We zullen ons nu eerst concentreren op de software-aspecten. We hebben hier opnieuw de probleemspecificatie nodig. Deze moet het probleem beschrijven. De invoer en uitvoer van het gewenste programma moet worden vastgelegd. Dikwijls treffen we ook de eis aan, dat de gewenste reactie van het programma binnen een bepaalde tijd ligt (de zgn. response-tijd). Tenslotte zijn er de eisen, die gesteld worden aan de maximale geheugenhoeveelheid, die door het programma mag worden gebruikt. In ons geval is dat zonder meer

duidelijk. De Teleac/DAI personal computer bevat een bepaalde hoeveelheid RAM geheugen (bij voorbeeld 8 K bytes). Het programma en de gegevens die u daarmee behandelt, moeten erin passen, anders kunt u het probleem niet oplossen.
Nu moeten we de *oplossingsmethode* kiezen. Een probleem kan op verschillende manieren worden opgelost. Gewoonlijk zal er een keuze gemaakt moeten worden uit verschillende *algorithmes* (zie *Microprocessors 1;* algorithme = oplossingsmethode), die tot de oplossing van het gestelde probleem leiden. Welke keuze er wordt gemaakt, zal worden bepaald door de mate, waarin de verschillende alternatieven tegemoet komen aan de eisen, die (ook voor wat betreft de computer) worden gesteld aan het programma. De criteria, zoals efficiënt gebruik van het geheugen, de reactiesnelheid van het programma, de begrijpelijkheid en de mogelijkheid om dat later aan te passen, zullen evenzeer meespelen in de argumentatie ten voordele van een bepaalde keuze.
Natuurlijk is het zo dat het kiezen van een bepaald algorithme niet zo maar kan gebeuren. Degene, die de keuze maakt, moet goed op de hoogte zijn met het algorithme zelf en met de programmering ervan. In snel toenemende mate komen er programma's op de markt, die werken en uitgetest zijn. In vele toepassingen spelen namelijk dezelfde soort problemen. Vandaar dat er nu ook bezien moet worden of er al programma's verkrijgbaar zijn, die aan de gestelde eisen (of bijna daaraan) voldoen.
Indien dit het geval is, kan men zich veelal tijd besparen. Het is dus echt niet zo dat men nu maar ieder algorithme zelf moet gaan programmeren. Zoals het in de hardware steeds minder noodzakelijk is apparatuur zelf te vervaardigen, zo tekent zich een zelfde ontwikkeling af in de software. Standaardpakketten software komen of zijn reeds verkrijgbaar. In de computerwereld spreekt men in zo'n geval wel van *plastic software.* Ook daaruit kan dus een keuze worden gemaakt. In het kader van dit boek is het evenwel de bedoeling, dat u als oefening zo veel mogelijk zelf programmeert. Bestaande bruikbare programma's kunnen dikwijls in een groter programma als subroutine worden ingebouwd.
Een ander aspect verdient daarbij vermelding: het is in dit licht gezien verstandig zelf geschreven programmatuur zo te ontwikkelen, dat het in een vergelijkbaar kader past. Programmeer zo mogelijk algemene software die later weer bruikbaar is. Dit stelt opnieuw eisen zoals een goede *definitie, algemeenheid* en goede *documentatie.*

9.4. Toestandsdiagram

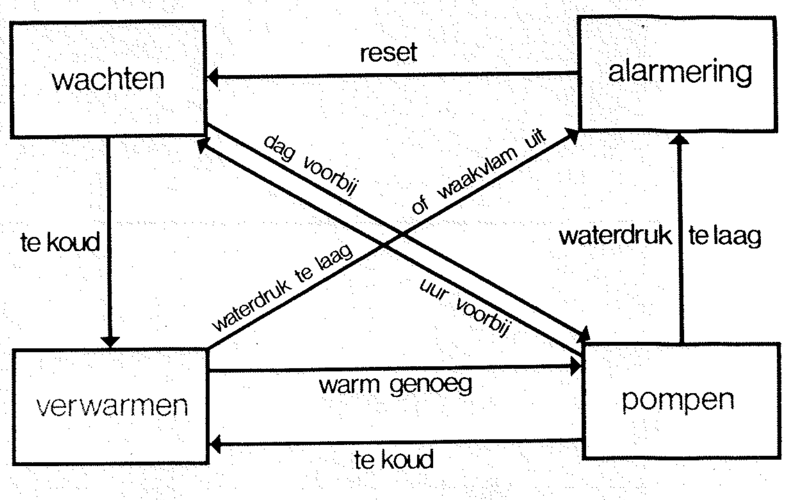
We worden dus ook op het software-niveau geconfronteerd met de vraag: kopen of zelf maken (zie afbeelding 9.3).
Bij de keuze van de oplossing is het nodig dat de oplossingsmethode zelf goed wordt beschreven. Een voortreffelijk hulpmiddel hierbij is het zgn. *toestandsdiagram,* dat wordt opgebouwd uit cirkels of rechthoeken met daartussen pijlen. Iedere toestand, waarin het programma zich kan bevinden, wordt door een *cirkel* weergegeven. De overgang van de ene toestand naar de andere wordt door middel van een *pijl* aangegeven. Het is nuttig om in de cirkels een naam te zetten, die de overeenkomstige toestand het best beschrijft. Bij de

pijlen dient de voorwaarde te staan waaronder de overgang wordt gemaakt.
Een toestandsdiagram dient men zuiver op te vatten als een hulpmiddel bij het bepalen van de gedachten. Het is verder van groot nut bij het opschrijven van de specificaties, omdat dan de gedachten beter zichtbaar kunnen worden gemaakt. Er bestaan geen stringente regels, waaraan een toestandsdiagram moet voldoen. De ervaring leert dat een eerste opzet met weinig toestanden en simpele overgangen wenselijk is. Iedere toestand op zich kan daarna nog verder gedetailleerd worden door middel van een nieuw toestandsdiagram, dat die ene toestand nauwkeuriger uitwerkt. Het is niet zo dat voor ieder programma een toestandsdiagram moet worden gemaakt. Bij een renteberekening bijvoorbeeld is er in feite slechts één toestand.



Afb. 9.3. Kiezen van het algorithme

Een toestandsdiagram heeft vooral zin als er achtereenvolgens acties moeten worden genomen. Dit komt vooral voor bij het meten en regelen van apparatuur, wat een belangrijk toepassingsgebied is voor microcomputers. (Soms worden de toestanden als rechthoeken aangegeven in plaats van cirkels.)
Een toestandsdiagram heeft verder nut wanneer later een programma *getest* wordt. Wanneer men begint met het programma te testen volgens de toestanden en overgangen, die op deze wijze op papier staan, geeft dit in eerste instantie een behoorlijke aanpak voor het testprobleem. Het diagram fungeert daarbij als een herkenningspunt van belangrijke programmadelen (zie afbeelding 9.4 dat een toestandsdiagram weergeeft van de verwarmingsregeling uit *Microprocessors 1*). Verder fungeert zo'n diagram als een uitgangspunt bij de opzet van het programma. De toestanden zijn hier, zoals bij *Microprocessors 1*, met rechthoeken aangegeven.
Voor de programma-opzet is het belangrijk het probleem op te splitsen in *deelproblemen*. Ieder deelprobleem kan weer in afzonderlijke stukken worden uitgesplitst. Dit uitsplitsen blijft zich herhalen tot-



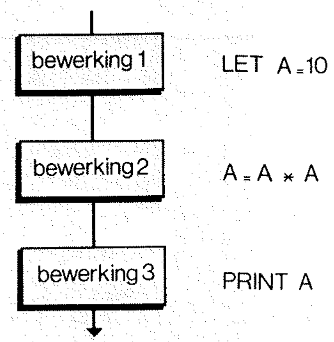
Afb. 9.4. Toestandsdiagram CV-regeling

dat herkenbare en eenvoudig programmeerbare eenheden overblijven. Op deze wijze blijft de programma-opzet overzichtelijk en onder controle. Herkenbare delen in verschillende deelproblemen, die dezelfde zijn, presenteren zich dan als natuurlijke kandidaten voor het maken van subroutines. Wanneer deelproblemen duidelijk genoeg geformuleerd zijn, kunnen ook meer mensen tegelijk aan het programmeerproces deelnemen.
Ieder kan dan de programmering van een deelprobleem op zich nemen, waarbij uiteraard voor duidelijke afspraken moeten worden gemaakt over de precieze aansluiting van de deelstukken (ook wel *interfacing* genoemd).

9.5. Stroomdiagram

Een belangrijk hulpmiddel bij het programmeren is het maken van stroomdiagrammen. We spreken van stroomdiagrammen, maar ook van *blokschema's*, *flowcharts* of *flowdiagrammen*. Hoe ze ook worden genoemd, het betreft een logisch opgebouwde verzameling van functionele blokken, die verbonden zijn door *lijnen*.
Elk blok heeft een symbolische betekenis en in elk blok kan een beschrijving worden gegeven van een of meer uit te voeren operaties. Deze operaties kunnen erg eenvoudig zijn (zoals tel 10 op bij variabele A) of meer gecompliceerd karakter hebben. In principe wordt een stroomdiagram van boven naar beneden en van links naar rechts gelezen. De lijnen tussen de blokken kunnen voorzien zijn van pijlen, waardoor dan een andere volgorde kan worden aangegeven. Niet elk stroomdiagram hoeft een gecompliceerde structuur te hebben. Dit is zelfs af te raden! Een eenvoudig stroomdiagram, waarin ieder blok een reeks handelingen vertegenwoordigt, is in eerste instantie te verkiezen boven een stroomdiagram, waarin elk detail is uitgewerkt. Dit laatste kan gebeuren in een later stadium, waarin de blokken van het aanvankelijk hoofdschema nader kunnen worden onderverdeeld. Daarbij zal een aanvankelijk als een blok opgegeven aantal handelingen telkens opnieuw worden verfijnd, zodat per blok een geringer aantal handelingen wordt opgegeven. Uiteindelijk kan dan een stroomdiagram worden verkregen, dat een één-op-één relatie vertoont met het nog te schrijven programma. Dat wil zeggen: ieder blok wordt later eenvoudig omgezet in statements. We zien ook

hier weer het principe van uitsplitsen en de herhaling daarvan. Dit principe is het leidende principe bij zowel het ontwerp van software als van hardware.

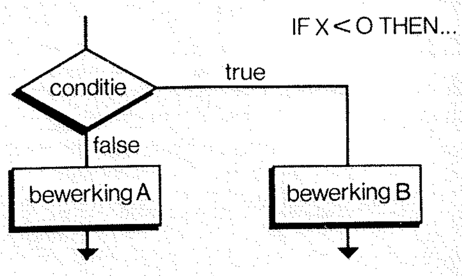


Afb. 9.5. Blokken uit een stroomdiagram

9.6. Structuren binnen een stroomdiagram

In dit gedeelte zullen we de *structuren* die we in voorgaande hoofdstukken reeds hebben gezien, nog eens herhalen. We voegen dan meteen de bijpassende *symbolen* uit de stroomdiagrammen toe. In feite is hier alleen dit laatste gedeeltelijk nieuw. Niettemin is het goed dat we de verschillende structuren nog eens grondig bezien. Zij vormen de *ruggegraat* van ieder programma.
Een stroomdiagram bestaat uit een aantal *blokken* (symbolen), die om te zetten zijn in een reeks opvolgende statements, zoals afbeelding 9.5 weergeeft. Deze statements kunnen rechtstreeks door de computer worden uitgevoerd. Bijvoorbeeld:

- het ophalen van data;
- het uitvoeren van een aantal bewerkingen op deze data, zoals optellen, aftrekken, enz.;
- uitvoer van deze data of invoer van nieuwe data.



Afb. 9.6. Conditionele sprong

Conditionele sprongen doorbreken de elkaar opvolgende rijen acties zoals die in afbeelding 9.6 zijn weergegeven en brengen een *vertakking* aan in het programma. Bij de uitvoering van een conditionele sprong wordt getest op de toestand, waarin een conditie zich bevindt. In BASIC zijn we ook in staat zo'n constructie weer te geven. Hiervoor kunnen verschillende BASIC expressies worden gebruikt.

De meest voor de hand liggende expressie is:

IF...THEN

Voorbeeld

IF X < 0 THEN PRINT 'X IS KLEINER DAN NUL'

Als X kleiner is dan nul, zal dit BASIC statement bij uitvoering ervan de tekst uitprinten 'X IS KLEINER DAN NUL'. Als X groter of gelijk is aan nul, wordt er niets geprint. In beide gevallen zal de executie van het op de volgende regel staande statement worden hervat. In ons voorbeeld is dus de conditie X < 0; de bewerking B, zoals in de afbeelding aangegeven, het printen van de genoemde tekst en bewerking A is niets.
Een andere mogelijkheid in BASIC is:

IF...GOTO

Voorbeeld

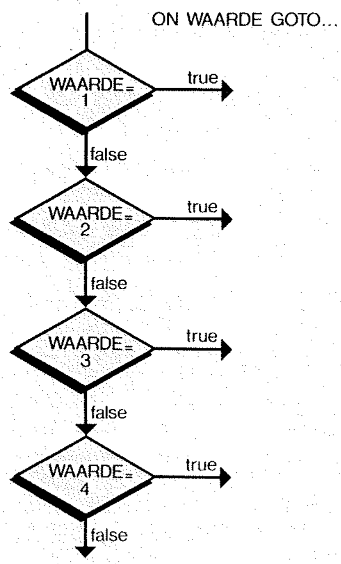
IF X < 0 GOTO 123

Als X kleiner is dan nul, zal BASIC regel 123 gaan uitvoeren. Daar staat dan bewerking B.
Indien X niet kleiner is dan nul, wordt normaal met de volgende regel vervolgd.
Daar hoort dan bewerking A te staan.
De IF . THEN en de IF . GOTO constructies zijn derhalve gelijkwaardig. Het verschil is dat THEN gevolgd moet worden door een

ander BASIC statement of een regelnummer, terwijl na GOTO een *regelnummer* behoort te komen. Het kan in een programma voorkomen dat onvoorwaardelijk moet worden gesprongen (men kan dit overigens even goed opvatten als een sprong met een voorwaarde, waaraan altijd wordt voldaan). Voor dit geval nu kent BASIC het statement GOTO.

Voorbeeld

GOTO 210



Afb. 9.7. Herhaald testen op een waarde

De BASIC regel, die na executie van bovenstaande regel wordt uitgevoerd, is de regel met het regelnummer 210.
Wanneer de bewerking B als een *subroutine* is geschreven, moet het BASIC statement in dit geval als volgt zijn:

GOSUB 210

De subroutine moet dan op regel 210 starten en moet nadien afgesloten worden met een RETURN statement zoals in hoofdstuk 6 is vermeld.
Wanneer het stroomdiagram een aantal achtereenvolgende tests bevat op een bepaalde waarde, kunnen ook andere BASIC statements worden gebruikt (zie afbeelding 9.7).
In de meeste computerboeken zal dit verschijnsel met een zogenaamd CASE statement worden aangeduid. Het is een *meervoudig* sprong.
In de eerste plaats is in BASIC mogelijk:

ON . GOTO

Voorbeeld

ON A GOTO 100, 200, 300, 500

Als A de waarde 1 heeft, wordt hierna het statement op regel 100 uitgevoerd.
Als A de waarde 2 heeft, wordt het statement op regel 200 uitgevoerd; als A de waarde 3 heeft wordt dat regel 300 en voor A is 4, wordt naar regel 500 gesprongen. Wanneer A niet een van deze waarden bezit, zal BASIC bij de uitvoering hiervan gewoonlijk een *foutmelding* geven.
Ook is toegestaan de constructie:

ON . GOSUB

Nu zal naar een subroutine worden gesprongen. Na terugkeer uit deze subroutine wordt het programma vervolgd op de regel volgend op ON. . .GOSUB.
Een veel voorkomende constructie in programma's is de sprong naar verschillende subroutines op grond van het teken of het nul zijn van een resultaat R.

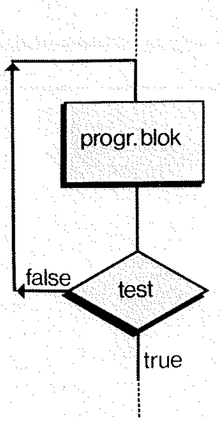
Voorbeeld

100 ON (SGN(R) + 2) GOSUB 200, 300, 400

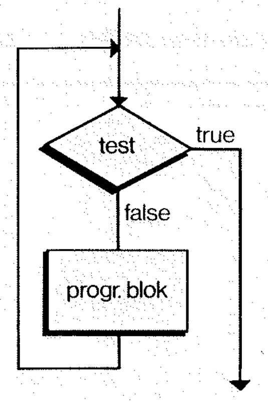
Het programma wordt vervolgd op regelnummer 200 als het teken van R *negatief* is, op 300 als R *nul* is en op 400 als R *positief* is.

Binnen subroutines zelf kunnen opnieuw subroutines worden aangeroepen. Men spreekt dan van *nesting* (= in elkaar vlechten) van

subroutines. Een drievoudige nesting betekent dat een subroutine aanroept. Deze tweede subroutine roept zelf weer een derde subroutine aan.
Op dezelfde wijze kan men *loops* (hieronder behandeld) binnen loops creëren. Ook dan wordt er gesproken van nesting van loops, of ook wel van 'geneste' loops.
Bij het gebruik van subroutines moet men er bijzondere aandacht aan besteden dat de subroutine altijd via een RETURN wordt verlaten. (Het terugkeeradres in de aanroepende routine wordt dan weer van de *stack* gehaald.) Zo moet ook altijd via een GOSUB naar de subroutine worden gegaan. (Het terugkeeradres wordt daarbij op de stack gezet.) Een belangrijk percentage aan fouten in programma's ontstaat doordat aan deze vereisten niet is voldaan. Er wordt dan ofwel met een gewone sprongopdracht (GOTO) naar de subroutine gegaan of er wordt gewoon doorgegaan in het programma. Het andere geval is het verlaten van de subroutine zonder een RETURN. Dit alles kan tot moeilijk te identificeren situaties aanleiding geven.
Ook worden conditionele sprongen opgenomen in zgn. *loops*. Een loop (of: lus) in een programma omvat een serie instructies, die telkens worden herhaald en die uit zichzelf de lus niet kunnen doorbreken. Een conditionele sprong, die in de lus is opgenomen, staat na de serie uit te voeren instructies, die het programmablok vormen. Een lus in een programma wordt meestal gebruikt om een bepaalde handeling een aantal malen te herhalen (zie afbeelding 9.8).
In de lus wordt bij voorbeeld een conditie bijgehouden. De conditionele sprong test of de conditie de juiste waarde heeft bereikt en springt uit de lus, wanneer dat het geval is.



Afb. 9.8. Herhaalde handeling test achteraf



Afb. 9.9. Herhaalde handeling, test vooraf

Het tweede gebruik van een lus-structuur betreft de test voordat (in plaats van nadat) de serie instructies is uitgevoerd. Er wordt telkens een aantal instructies uitgevoerd, die een bepaalde toestand niet verandert. Zodra de vereiste toestandsverandering optreedt, wordt de lus verlaten. Dit gebeurt met behulp van de *conditionele* sprong, die dan voor de serie instructies staat, die het programmablok vormt. In dit geval is het dus mogelijk dat de lus in het geheel niet wordt doorlopen. Bij de test achteraf wordt de lus minstens eenmaal doorlopen (zie afbeelding 9.9).

Wanneer van tevoren het aantal keren dat een opdracht moet worden herhaald, berekend kan worden, is het volgende BASIC statement bruikbaar:

FOR . TO

Zoals in hoofdstuk 5 reeds is vermeld moet zo'n loop worden afgesloten met een NEXT statement.

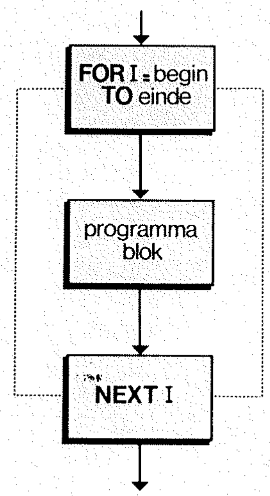
Voorbeeld

```
*LIST
10     FOR I=2.0 TO 11.0
20     PRINT I
30     NEXT I
*RUN
 2.0
 3.0
 4.0
 5.0
 6.0
 7.0
 8.0
 9.0
 10.0
 11.0
*
```

De loop wordt nu 10 keer doorlopen. De waarden 2 tot en met 11 worden onder elkaar uitgeprint.
Wanneer de stapgrootte van I niet 1 is, moet nog worden vermeld hoe groot de stappen zijn, met behulp van STEP.

Voorbeeld

```
*LIST
10     FOR I=2.0 TO 11.0 STEP 4.0
20     PRINT I
30     NEXT I
*RUN
 2.0
 6.0
 10.0
*
```



Afb. 9.10. FOR/NEXT

De loop wordt nu driemaal uitgevoerd. De waarden 2, 6 en 10 worden onder elkaar geprint. We zien hier dus dat we de *basisblokken* van een stroomdiagram gemakkelijk in herkenbare BASIC statements kunnen omzetten.
We hebben hier een speciaal symbool voor: FOR NEXT zoals afbeelding 9.10 weergeeft. Let er bij FOR NEXT en GOSUB op dat het aantal keren dat er 'genest' kan worden in elke computer beperkt is. Raadpleeg weer uw handleiding.
Er bestaan ook alternatieven voor het gebruik van stroomdiagrammen. Voorbeelden hiervan zijn *Nassi Schneidermann-diagrammen* en *decision tables*, die hier niet nader worden verklaard. Ook daar treden grafische weergaven van de opbouw van het programma op.

Gewoonlijk is het gebruik ervan wat moeilijker. De voordelen liggen in de grotere gestructureerdheid, die bereikt kan worden. Voor een boek als dit is gekozen voor de meer gebruikte stroomdiagrammethode. De lezer is dan goed in staat die vormen te begrijpen en zelf te hanteren. Voor meer gevorderde programmeurs is het raadzaam ook andere methodes te bestuderen en te gebruiken.

9.7. Programmeerregels

Er zijn enkele regels, die het de programmeur gemakkelijk maken om een duidelijk systematisch en correct programma te schrijven. Wanneer men tijdens het programmeren deze regels in acht neemt, wordt de kans op een begrijpelijk en correct programma aanzienlijk vergroot.
Met behulp van eenvoudige en bekende structuren bouwt men een programma op. De belangrijkste regels zijn:

- Ieder deelprogramma mag slechts één ingang en één uitgang bezitten. In dat deelprogramma wordt altijd op dezelfde plaats begonnen. Het deelprogramma wordt altijd op dezelfde plaats verlaten. Een sprong naar een andere plaats in de routine is derhalve niet toegestaan. Evenmin is het toegestaan om op meer plaatsen uit de routine te springen. Een opzet, die op deze leest is geschoeid, maakt programma's doorzichtiger en de kans op fouten wordt drastisch verkleind.
- Ieder deelprogramma (of routine) mag slechts uit een of meer opdrachten worden samengesteld volgens de volgende basisvormen:

1. *Een enkelvoudige opdracht.* In BASIC kan men hieronder een statement verstaan, dat geen sprong bevat.

Voorbeeld

```
LET A = 10
```

2. *Een samengestelde opdracht.* Een samengestelde opdracht is een aaneenschakeling van enkelvoudige opdrachten.

3. *Een selectie-opdracht.* In een selectie-opdracht wordt op grond van een conditie een beslissing gemaakt. Eerder in dit hoofdstuk is al beschreven dat BASIC daarvoor de statements kent:

```
IF . THEN
IF . GOTO
GOTO
GOSUB
ON . GOTO
ON . GOSUB
```

4. *De herhalingsopdracht.* De herhalingsopdracht geeft aan dat herhaaldelijk door een loop wordt gegaan. In feite hebben we hier te maken met een combinatie van een enkelvoudige opdracht en een sprong. In principe is ieder programma dan op te bouwen uit uitsluitend enkelvoudige en selectie-opdrachten. De herhaling kan met een van de bovenstaande

BASIC instructies worden gedaan of met:

FOR . TO . STEP, gevolgd door NEXT

Maak zo weinig mogelijk gebruik van het onconditionele GOTO statement (de onconditionele sprong). Bij gebruik van dit statement worden programma's gewoonlijk ondoorzichtig en moeilijk testbaar. In BASIC is het GOTO statement vaak evenwel noodzakelijk bij een beslissing in bijvoorbeeld de volgende constructie:

70 IF A > 25 GOTO 110

80 GOTO 190

In feite zou het hier prettiger zijn geweest om meteen te programmeren:

70 IF (A > 25) THEN 110 ELSE 190

Dus als A > 25, verder gaan op regel 110; zo niet, dan verder gaan op regel 190.

Veel talen staan deze laatste constructie toe. Sommige software deskundigen gaan dan ook zover dat ze stellen dat in zo'n geval het gebruik van het onconditionele GOTO statement geheel verboden zou moeten worden.

Beperkingen van BASIC
Wanneer we nu toch over beperkingen van BASIC praten, is het goed om een 'gebrek' in BASIC te signaleren. BASIC kent niet het begrip 'label' zoals dat bijvoorbeeld in de ASSEMBLER taal (zie *Microprocessors 1* en in dit boek hoofdstuk 12 en volgende hoofdstukken) en in andere talen wel is toegestaan. Het gebruik van labels heeft als voordeel dat een sprong naar een routine (GOTO) of subroutine (GOSUB) reeds kan worden aangegeven als het juiste beginadres van de routine nog niet bekend is. De vertaler (zelf ook een computerprogrammeur) voor een computertaal, waarin labels zijn toegestaan, vertaalt dan dit label in het eigenlijke *beginadres*. Bij BASIC moet de programmeur dit zelf doen. Een tweede voordeel van het gebruik van labels is dat de labelnaam een duidelijke betekenis kan hebben, bijvoorbeeld FOUT, GOED, OPNIEUW, VOORUIT, enz. Het programma wordt daarmee leesbaarder. In BASIC moeten we met het REM statement leesbaarheid creëren.
Het is een goede gewoonte tijdens het opschrijven van een BASIC programma in eerste instantie ook labels te gebruiken. Voordat het programma in de computer wordt ingetypt kan de programmeur ze dan zelf in regelnummers omzetten. Het feit dat een numerieke variabele die nog geen waarde heeft gekregen door BASIC als de waarde 0 wordt beschouwd kan men ook als een 'gebrek' aanmerken. De meeste andere talen geven in dit geval een *foutmelding*. Wenst men namelijk een 0 dan is het beter dit duidelijk te vermelden als toekenning. Voor de string variabele geldt dat BASIC deze in eerste instantie als blanco beschouwt.
In de loop van het schrijven van programma's op papier ontstaat er al snel een verstoring in het patroon van de met 10 oplopende regelnummers. Het is eveneens een goede gewoonte die regelnummering weer in orde te brengen voordat het programma wordt ingetypt. Sommige computers hebben voorzieningen voor een automatische

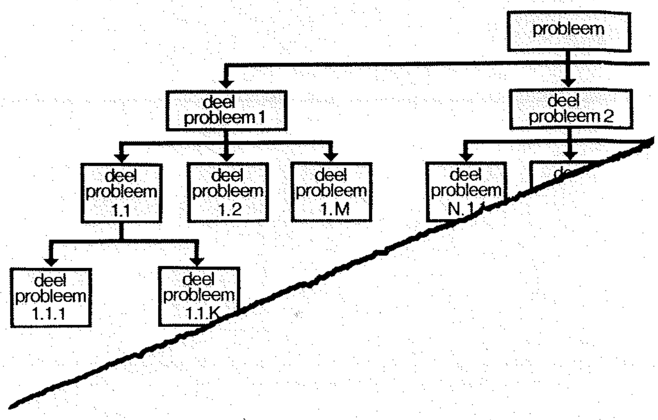
regelnummering c.q. hernummering. De Teleac/DAI personal computer kent deze voorziening niet.
Het is van groot belang het programma eerst op papier goed in orde te brengen en te doordenken. Pas daarna moet het programma ingetypt worden. Vervolgens wordt het programma getest. Komen daarbij fouten te voorschijn, analyseer dan die fouten weer achter het bureau. Probeer niet op de computer zelf snel even allerlei verbeteringen aan te brengen. Doordenk het programma steeds eerst opnieuw!
De methode om steeds programma's te bedenken, in te typen en direct op de computer te testen, leidt in de praktijk vrijwel altijd tot ongestructureerde, slechte programma's. De gevolgen van dwang moeten hierbij niet onderschat worden.
Als men achter de computer zit te programmeren voelt men zich gewoonlijk gedwongen snel te werken. Dit lijkt een soort dwang van de computer te zijn om wachten te voorkomen. Vrijwel niemand ontkomt daar aan. Voorkom deze situatie door goed vooruit te denken.

Dus: eerst denken voordat u met de computer aan het werk gaat (de bekende spreuk: *Bezint eer gij begint).*

Top-down design
Wanneer een deelprogramma volgens bovenstaande regels is geprogrammeerd, kan het weer in een groter geheel worden opgenomen. In dit grotere geheel kan men dan zo'n *deelprogramma* weer als een enkelvoudige opdracht opnemen.
In feite kan men de bovenstaande werkwijze met vier basiselementen vergelijken met het bouwen met een blokkendoos, bestaande uit eenvoudige blokken. Hoewel elk blok simpel is, is er een grote verscheidenheid aan objecten mee te construeren. In softwaretermen vertaald betekent het, dat er programma's in grote verscheidenheid kunnen worden gemaakt. De programmeur heeft uiteraard de vrijheid om naar keuze zijn deelroutines te kiezen. In dat proces is ook een duidelijke leidraad aan te geven. Een verstandige aanpak gaat uit van zogenaamd *top-down design*. De programmeur begint zijn probleem in delen te splitsen. Eerst grote herkenbare delen, die een bepaalde functie verrichten. Daarna wordt iedere functie op dezelfde manier in kleinere delen gesplitst. Bij het oplossen van een probleem heeft men vrijwel altijd al een idee welke delen zullen ontstaan. De programmeur kan zich het best laten leiden door zijn gezond verstand. Ook hier kan het gezond verstand aangevuld worden met meer wetenschappelijke methodes.
Een belangrijke methode is die van *Information Hiding* (ofwel informatie verstoppen). Deze dwingt een programmadeel alleen die informatie te gebruiken, die strikt nodig is. Alle andere routines mogen die informatie alleen maar via dit programmadeel verkrijgen. Dit komt wonderwel overeen met het consequent toepassen van 'gezond verstand'.

Bij het opdelen van het probleem in deelproblemen kan het programma heel goed zo geconstrueerd worden dat ieder deelprogramma een deelprobleem oplost. Ieder deelprogramma heeft zo een duidelijke functie, die eenvoudig omschreven moet zijn.

Dit gaat zo door totdat de programmadelen zo klein zijn dat ze eenvoudig direct kunnen worden geprogrammeerd (zie afbeelding 9.11).

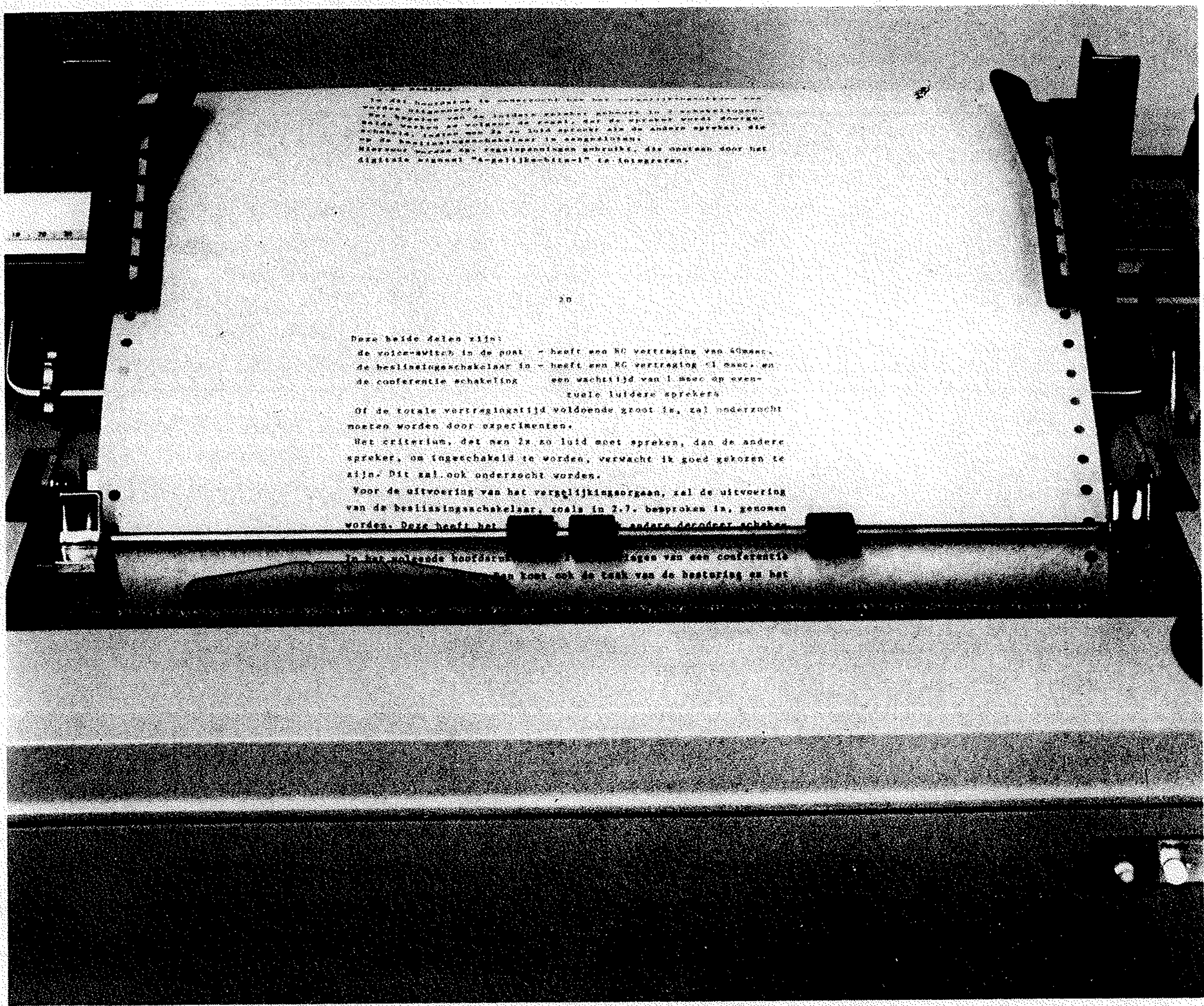


Afb. 9.11. Top-down ontwerp

Het verdient aanbeveling om in het programma zelf voldoende *commentaar* aan te brengen. Op die manier is zo'n programma ook voor een buitenstaander begrijpelijk. Men moet zich realiseren dat de programmeur zelf na enige tijd als een buitenstaander kan worden beschouwd. Hij kent dan namelijk niet meer precies alle details van het programma. Een verstandige opzet en voldoende commentaar moet ook voor hem voldoende zijn om zich weer in te werken.
Het is nuttig vanaf het begin met de gegeven aanwijzingen rekening te houden.
Voor velen zal in dit stadium het begrijpen van en het werken met de BASIC statements nog een groot probleem zijn. In een latere fase wordt dat echter beheerst.
Hier kan men een parallel trekken met *tennislessen*. In eerste instantie heeft een cursist al de grootste moeite om de bal te raken. Voor hem is dat op dat moment het *kernprobleem*. Een goede tennisleraar wijst hem er meteen op dat de slaguitvoering belangrijker is dan het raken van de bal. Een verkeerde slagbeweging later corrigeren is bijna een onmogelijkheid. Het raken van de bal komt na enige oefening vanzelf.
In ons geval komt de bekendheid met de BASIC statements na enige oefening ook vanzelf. Het beste kan dit op een computer geoefend worden. Maar het goed structureren van een programma is iets, waar u steeds opnieuw aan moet denken.
Op dit punt moet u altijd uw programma's kritisch blijven bezien. Het tonen van uw programma's aan anderen en ze er mee laten werken kan u een goede terugmelding geven.

9.8. Testen van programma's

Het testen van de programma's kan in een *bottom-up* (= van beneden naar boven) aanpak plaatsvinden. Eerst worden de elementaire routines getest. Daarna worden ze samengevoegd tot een groter geheel. Dat grotere geheel wordt dan weer getest. Vervolgens wordt in een aantal stappen van samenvoeging en testen het correct werkend totaal bereikt.
Bij het testen van een routine kan het best voor de volgende aanpak



De microcomputer maakt het rapporteren tot een eenvoudige zaak. De tekst wordt door de computer zo bewerkt dat links een mooie kantlijn ontstaat. (Tekstverwerkingssysteem Vakgroep EB, TH Eindhoven)

gekozen worden: test de routine eerst voor *bekende* waarden en een bekend resultaat. Daarna moet er voor *extreme* en *kritische* waarden getest worden. Als de routine voor al deze waarden goed werkt, is de kans groot dat hij als geheel ook goed zal werken.
Voor eenvoudige routines is het soms mogelijk te testen voor alle waarden die kunnen optreden. Dit heet een *uitputtende* test. Wanneer het aantal mogelijke waarden groot wordt, duurt het uitputtend testen te lang en moet op bovengenoemde methoden worden overgegaan.
Er wordt door veel onderzoekers gezocht naar methodieken om de juistheid van programma's via *wiskundige* bewijzen aan te tonen. Voor sommige routines is zo'n bewijs reeds mogelijk. Voor meer gecompliceerde, lange programma's is dit nog ondoenlijk. Vandaar dat het op verstandige wijze in *modules* verdelen vooralsnog de meest bruikbare methode is. Voor de opzet van programma's kan men het best alle deelroutines definiëren. Iedere subroutine kan dan afzonderlijk worden getest.

Ook de *top-down* aanpak van testen kan hier zinvol werken. De gegevens voor de deelroutinetests kunnen op deze wijze vaak eenvoudig worden verkregen. Daardoor kan een tijdsbesparing optreden.
In het algemeen verwacht een subroutine invoergegevens en zij levert ook uitvoergegevens af. De manier waarop deze gegevens en hun waarden worden behandeld vormen een kritisch onderdeel van de programma-opzet. Hier moet bij het testen extra aandacht aan worden besteed.
Het testen van programma's mag zeker niet worden onderschat. Cijfers geven aan dat de testfase van programma's wel 40% van alle aan een probleem bestede tijd in beslag kan nemen. Voor de beginner is ook dit een fase waarin met verstand, een goede programma-opzet en oefening veel kan worden bereikt. Faciliteiten, die in BASIC voor het testen gebruikt kunnen worden, zijn in hoofdstuk 4 behandeld.
De testfase is verspreid over alle ontwerpfasen. Steeds moet er een test op programma's of gedachten plaatsvinden.
De afwezigheid van fouten tijdens het testen is nog geen bewijs dat het programma correct is! Bij nieuwe tests kunnen toch nog fouten aan het licht komen. Een goede test moet dan ook alle te verwachten situaties omvatten. Uit tijdsoverwegingen is dit soms niet haalbaar. Dan zal er een verstandige beperking aangebracht moeten worden.

9.9. Overdracht van gegevens

Er zijn veel manieren waarop deze gegevens van de ene naar de andere routine kunnen worden overgedragen.
Deze overdracht staat in de vakliteratuur bekend onder de naam *parameter passing*.
Er bestaan drie methoden voor in machinetaal geprogrammeerde routines. (Het gebruik van de *machinetaal* is in *Microprocessors 1* uitgelegd.)

- De overdracht vindt via vast afgesproken geheugenplaatsen plaats. Iedere routine haalt daar zijn eigen gegevens vandaan en zet daar ook zijn resultaten neer.
- De gegevens worden via de stack, ofwel stapel, doorgegeven. Het aanroepende programma zet de gegevens voor de aan te roepen routine op de stack en haalt daarvan de resultaten weer op. Voorwaarde is dat de subroutine precies het juiste aantal keren naar de stack gaat, zodat na afloop het terugkeeradres weer boven op de stack aanwezig is. De stack bewaart het adres, waarnaar de subroutine moet terugkeren. Het begrip 'stack' en de betekenis daarvan voor subroutines is in *Microprocessors 1* uitgelegd en daar vergeleken met de stapel borden in een cafetaria.
- Het gebruik van *microprocessorregisters*. Register A kan bijvoorbeeld altijd worden gebruikt om een resulterend byte over te dragen. Voorwaarde hier is dat het opgeroepen programma geen andere dan de afgesproken registers van waarde doet veranderen. Wanneer dat niet mogelijk is, moet duidelijk aangegeven worden welke registers worden beïnvloed en welke niet. Soms is het dan nodig de registers, die veranderd worden, eerst als het ware te 'redden', dat wil zeggen dat de inhoud ervan voor een later gebruik op een afgesproken geheugenplaats wordt neergezet. Hiervoor kan uiteraard ook de stack worden gebruikt. In BASIC komt

het begrip register niet voor. Overdracht via registers is dientengevolge bij de BASIC uitgesloten. In *Microprocessors 1* is uitgebreid op registers ingegaan.

Het is mogelijk in een groot programma van alle drie bovengenoemde methodes gebruik te maken en ze te combineren. Voor de eenvoud van de programmering is het echter beter slechts één methode te kiezen en daar dan ook consequent aan vast te houden. Voor de documentatie kan het best voor iedere subroutine dezelfde opzet worden gekozen. Eerst een *kop,* waarin de naam van de subroutine staat. Vermeld daarbij wat de routine doet, wat hij aan invoer nodig heeft, wat de uitvoer is en wat de datum van de huidige versie is. Verder moet worden aangegeven welke andere routines door deze subroutine worden aangeroepen. Uiteraard doen de hierboven genoemde vereisten zich meer gelden naarmate de programma's groter worden.

In BASIC kunnen we voor de overdracht van de gegevens twee methoden onderscheiden.

- Een variabele naam geeft het mechanisme van de overdracht weer. Voorbeeld: in het hoofdprogramma krijgt de variabele A een waarde. De subroutine gebruikt de variabele A als inputgegeven in een BASIC statement. In feite dragen we dan de gegevens over, terwijl we daarbij gebruik maken van de BASIC interpretor, die in ons Teleac/DAI personal computersysteem aanwezig is.
- De overdracht via vast afgesproken geheugenplaatsen. De BASIC functies PEEK en POKE kunnen daartoe worden gebruikt. Deze functies lezen van en schrijven naar geheugenplaatsen, zoals in hoofdstuk 17 nader wordt uitgelegd.

De programma-opbouw in BASIC vereist een zorgvuldige *specificatie.* Dan is een verstandige en logische verdeling in subroutines noodzakelijk. Iedere subroutine wordt opgebouwd uit een beperkt aantal programmaconstructies. De tot nu toe gegeven aanwijzingen dienen slechts als aanwijzing. Zij zullen des te nadrukkelijker spreken, wanneer u enige ervaring hebt verkregen met het maken van grotere programma's. Voor kleinere programma's zijn de voordelen minder duidelijk zichtbaar. Ons boek bevat voornamelijk kleine programma's, zodat daarin de gegeven methode niet altijd kan worden gedemonstreerd.

9.10. Regels voor BASIC programma's

We willen echter niet nalaten hier ook wat meer stringente regels voor het schrijven van BASIC programma's te geven, waarmee u reeds bij kleinere programma's uw voordeel kunt doen. Deze regels zijn:

- Gebruik veel REM statements om in het programma voldoende duidelijk aan te geven wat u precies doet.
- Begin het programma altijd met een commentaar-kop, die de betekenis van het programma aangeeft.
- Kies variabele namen met een duidelijk sprekende betekenis. *Opmerking:* we zullen later zien dat de Teleac/DAI personal

computer namen van variabelen toestaat tot 14 characters lengte. Bij andere computers is dit vaak minder.

- Begin de regelnummers in tientallen op te hogen, dus 10, 20, enz. Op deze wijze kunt u later eenvoudig nog statements tussenvoegen.
- Zet slechts één statement op één regel, dan blijft uw programma overzichtelijk.
- Gebruik steeds dezelfde variabele namen voor dezelfde begrippen. Dus voor variabelen, die als teller of als index worden gebruikt, kiest u altijd de namen I%, J%, K% of I, J en K.
- Gebruik nooit één variabele naam in verschillende delen van het programma voor verschillende doeleinden. Anders ontstaat beslist begripsverwarring.
- Gebruik subroutines met een duidelijke en eenvoudige functie. Geef met commentaar de te verrichten functie, de benodigde invoergegevens en het af te leveren resultaat aan. Verlaat de subroutine uitsluitend via een RETURN statement.
- Zet subroutines bij elkaar voor de DATA statements.
- Zet DATA statements bij elkaar aan het eind van het programma.
- Zet DIM statements bij elkaar aan het begin van het programma.

Op de boven aangegeven wijze kunt u een overzichtelijk en duidelijk programma produceren. Het programma zelf vervult dan ook in belangrijke mate de rol van documentatie. Het programma zelf, de gebruikershandleiding en de flowcharts tezamen vormen de totaal benodigde documentatie. Het is zeer verstandig de overwegingen die tot bepaalde keuzes in de specificaties en de programmering hebben geleid, nog apart te beschrijven en aan de documentatie toe te voegen.

> In de Teleac/DAI-listings zetten we vaak INPUT en PRINT alleen bij elkaar op één regel om het verschil in INPUT met de gebruikelijke methode te benadrukken.

9.11. Het besparen van geheugenruimte

Er kunnen zich echter problemen voordoen omdat op deze wijze te veel geheugenruimte in beslag wordt genomen. Als u geheugenruimte wilt besparen, kunt u de volgende maatregelen overwegen:

- Gebruik meer statements op één regel. Daardoor bespaart u op de interne opslag van regelnummers.
- Gebruik waar mogelijk *integer* variabelen in plaats van *floating point* variabelen.
  Deze laatste nemen namelijk meer ruimte in, dus A% in plaats van A gebruiken (zie hoofdstuk 2).
- Verwijder zoveel mogelijk *commentaar*. U kunt het commentaar wel in uw geschreven origineel bijschrijven, zodat u later toch de beschikking over informatie hebt.
- Vermijd overbodige blanco's.
- Ken een waarde, die meermalen wordt gebruikt, toe aan een variabele. Waar eerst in het programma een constante zou staan, staat nu een variabele. In het opgeslagen programma hoeft dan niet steeds opnieuw ruimte voor de constante te worden gereserveerd.
- Gebruik dezelfde variabele naam zoveel mogelijk, ook als daaraan verschillende, niet conflicterende betekenissen moet worden gehecht. Let er dan zelf wel steeds op wat de betekenis op ieder moment is.

- Maak de arrays niet groter dan per se noodzakelijk is.
- Gebruik overal waar dezelfde statementvolgorde voorkomt, een subroutine in plaats van een herhaling van de statements.
- Gebruik het 'nulde' element van arrays, bijvoorbeeld A(0), B$(0), C (0,Y) enz.
- Laat het END statement weg als dit niet noodzakelijk is in uw computer.
- Gebruik zo kort mogelijke variabele namen, liefst één letter.

Het is duidelijk dat de maatregelen voor het in ruimte verkleinen van het programma veelal in tegenspraak zijn met de eerst gegeven algemene regels.
Het is dan ook aanbevelenswaardig eerst van de verstandige regels uit te gaan.
Pas wanneer ruimtegebrek optreedt, kan men dan maatregelen nemen. Het is dan nuttig het via de regels geschreven programma als extra documentatie te hanteren.
Vooral voor beginners verdient het grote aanbeveling de *doorzichtigheid* van programma's zo groot mogelijk te maken. Het zich bekommeren om geheugenruimte komt pas op de tweede plaats. Geheugenuitbreiding is tegenwoordig goedkoop. Zo kan ook een oplossing worden gevonden door middel van aankoop van geheugen.

9.12. Executiesnelheid verhogen

Een ander voorkomend probleem is dat het programma te langzaam wordt in executie. Dan zijn er weer andere, opnieuw veelal *contrasterende* maatregelen nodig:

- Gebruik zo weinig mogelijk subroutines. GOSUB werkt vertragend.
- Ook sprongen door middel van GOTO werken vertragend. Het gebruik daarvan wordt hier dan ook afgeraden.
- FOR . NEXT en IF . GOTO loops zijn wat resultaat betreft gelijkwaardig. Wanneer de ene vorm duidelijk sneller is dan de andere, verdient de snellere natuurlijk de voorkeur. In het geval van de Teleac/DAI personal computer is dit duidelijk de FOR . NEXT loop. Deze laatste bevordert bovendien nog de doorzichtigheid van de programmering.
- Integer-berekeningen gaan sneller dan floating point berekeningen. Gebruik als het even kan daarom integers. Vooral voor de telling in een loop verdient gebruik van integer variabelen de voorkeur.
- Ingewikkelder operaties als vermenigvuldigen, delen, enz. zijn in executie langzamer dan eenvoudige operaties als optellen. Een uitdrukking zoals 2 * K is duidelijk langzamer in executie dan K + K. Ga dus na of dit soort omzettingen mogelijk is. Vooral de machtsverheffing, zoals A↑ 3, is tijdrovend. Het is aan te bevelen dit te vermijden. Dit kan hier bijvoorbeeld door de uitdrukking A * A * A te gebruiken.
- Soms komen in reeksen expressies gelijke constructies voor. Dat wil zeggen dat een rij statements met hetzelfde resultaat op meerdere plaatsen voorkomt. Wanneer we nu dit resultaat één keer aan een variabele toewijzen kan de waarde van deze variabele verder gewoon gebruikt worden. De berekening hoeft dan niet steeds opnieuw gedaan te worden.

- Van buitenaf aangeboden signalen kunnen worden getest met een IF . THEN combinatie. Het in hoofdstuk 17 te behandelen WAIT statement kan in dit geval hetzelfde doen, maar het werkt sneller.
- Gebruik variabelen in plaats van constanten. Transformatie van een constante naar bijvoorbeeld drijvende komma formaat kost gewoonlijk meer tijd dan het halen van een variabele.
- Laat REM statements en overbodige blanco's weg. De interpreter heeft tijd nodig om te herkennen dat hij deze delen mag overslaan.

Ook hier geldt dat de meeste maatregelen als *noodmaatregelen* moeten worden beschouwd. Maar het is nuttig om te weten hoe te handelen als ruimte of tijdgebrek optreedt.

Samenvatting hoofdstuk 9

We hebben in dit hoofdstuk vele stappen, die noodzakelijk zijn in het programmeerproces, behandeld. De noodzaak van een duidelijke specificatie is benadrukt.
Het verband tussen een stroomdiagram en BASIC statements is gelegd.
Tenslotte hebben we algemene regels voor de opzet van BASIC programma's gegeven. Deze regels zijn nog eens toegelicht voor het geval er ruimte of tijdnood ontstaat in een BASIC programma.

Multiple choice-vragen bij hoofdstuk 9

A. **a.** Een toestandsdiagram is een hulpmiddel bij de probleemspecificatie.
b. Vooronderzoek behelst de keuze uit verschillende algorithmes.

1 uitspraak a is goed en uitspraak b is goed
2 uitspraak a is fout en uitspraak b is goed
3 uitspraak a is goed en uitspraak b is fout
4 uitspraak a is fout en uitspraak b is fout

B. Welke van de hier genoemde eisen hoort niet in de probleemspecificatie thuis?

1 invoer-eisen
2 uitvoer-eisen
3 begrijpelijkheid van het programma
4 respons-tijd

C. Bij welke van de genoemde onderwerpen fungeert een toestandsdiagram niet als hulpmiddel?

1 testen van programma's
2 uitwerken van het blokdiagram
3 bij het ontleden van het probleem
4 formuleren van de gedachten

D. **a.** Een CASE statement kan in BASIC worden weergegeven door ON. GOTO.
b. Een voorwaardelijke sprong wordt in BASIC weergegeven door het GOTO of GOSUB statement.

1 uitspraak a is goed en uitspraak b is goed
2 uitspraak a is fout en uitspraak b is goed
3 uitspraak a is goed en uitspraak b is fout
4 uitspraak a is fout en uitspraak b is fout

E. **a.** Bij subroutines moet men er op letten dat de aanroep via GOTO verloopt en de terugkeer via RETURN.
b. FOR. TO wordt o.a. gebruikt voor een lus als het aantal malen dat de lus moet worden doorlopen bij het ingaan van de lus bekend is.

1 uitspraak a is goed en uitspraak b is goed
2 uitspraak a is fout en uitspraak b is goed
3 uitspraak a is goed en uitspraak b is fout
4 uitspraak a is fout en uitspraak b is fout

F. **a.** In BASIC kunnen gegevens van het ene naar het andere programma worden overgedragen via het geheugen.
b. Het overdragen van gegevens naar subroutines wordt wel genoemd: top-to-down aanpak.

1 uitspraak a is goed en uitspraak b is goed
2 uitspraak a is fout en uitspraak b is goed
3 uitspraak a is goed en uitspraak b is fout
4 uitspraak a is fout en uitspraak b is fout

G. Waar moet bij het programmeren in eerste instantie de meeste aandacht aan worden besteed?

1 besparing geheugenruimte
2 verhoging executiesnelheid
3 doorzichtigheid
4 gebruik van REM statements

H. **a.** Onvolledige specificaties kunnen leiden tot foutieve programma's.
b. In de evaluatiefase wordt o.a. naar de mate van tevredenheid gekeken van de gebruiker.

1 uitspraak a is goed en uitspraak b is goed
2 uitspraak a is fout en uitspraak b is goed
3 uitspraak a is goed en uitspraak b is fout
4 uitspraak a is fout en uitspraak b is fout

I. **a.** Bij de programma-opzet is het verstandig om in het eerste stroomdiagram zoveel mogelijk details mee te nemen.
b. Veelvuldig gebruik van GOTO statements maakt een programma ondoorzichtig.

1 uitspraak a is goed en uitspraak b is goed
2 uitspraak a is fout en uitspraak b is goed
3 uitspraak a is goed en uitspraak b is fout
4 uitspraak a is fout en uitspraak b is fout

J. **a.** Het opzetten van een programma kan in eerste instantie het best met de computer zelf plaatsvinden, zodat fouten direct worden gevonden.
b. De correctheid van een programma kan altijd worden aangetoond met uitgebreide programmatests.

1 uitspraak a is goed en uitspraak b is goed
2 uitspraak a is fout en uitspraak b is goed
3 uitspraak a is goed en uitspraak b is fout
4 uitspraak a is fout en uitspraak b is fout

Hoofdstuk 10

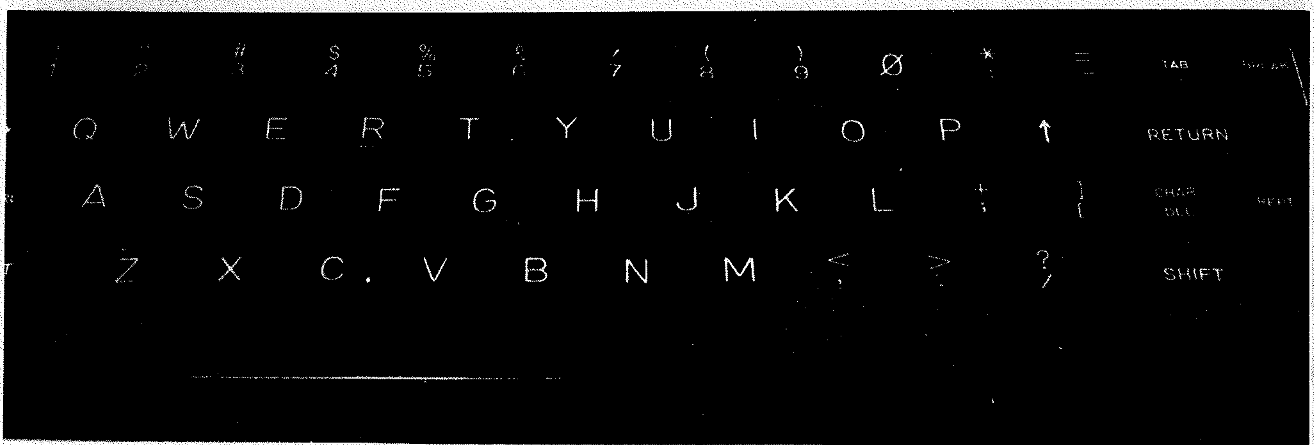
Tekst in BASIC

In dit hoofdstuk zien we hoe we met tekst in BASIC kunnen werken. Daarop kunnen we namelijk ook bewerkingen doen. Veel van de in de vorige hoofdstukken genoemde statements kunnen op tekst worden toegepast. Er zijn speciale functies gedefinieerd, die alleen op tekst opereren om gedeelten uit die tekst te selecteren en te bewerken. In dit hoofdstuk zien we verder hoe getallen en tekst in elkaar kunnen worden overgezet. Tenslotte worden de tekstopmaakfuncties behandeld.

10.1. Inleiding

Wanneer we een tekst intypen zoals 'IK VIND JE AARDIG', dan zijn we duidelijk niet met het intypen van getallen bezig, maar met het intypen van tekst. Zo'n stuk tekst wordt in BASIC begrippen een *string* genoemd (string betekent: een *lint* of *koord).*
Een string bestaat uit een rij tekens, die door de computer niet als getallen, maar op een andere wijze wordt behandeld. Deze tekens worden wel *alfanumerieke tekens* genoemd. Zij bevatten namelijk letters uit het alfabet (alfanumerieke informatie) en ook cijfers (numerieke informatie). Het is zelfs zo dat hierin ook tekens voorkomen, die niet kunnen worden geprint maar alleen op andere wijze worden verwerkt (bijvoorbeeld het laten gaan van een *belsignaal*). Al deze tekens kunnen we via het toetsenbord invoeren (zie afbeelding 10.1).

Afb. 10.1. Toetsenbord Teleac/DAI personal computer



In feite is tekst in BASIC natuurlijk niet iets nieuws. In de vorige hoofdstukken hebben we bijvoorbeeld al de uitdrukking gezien:

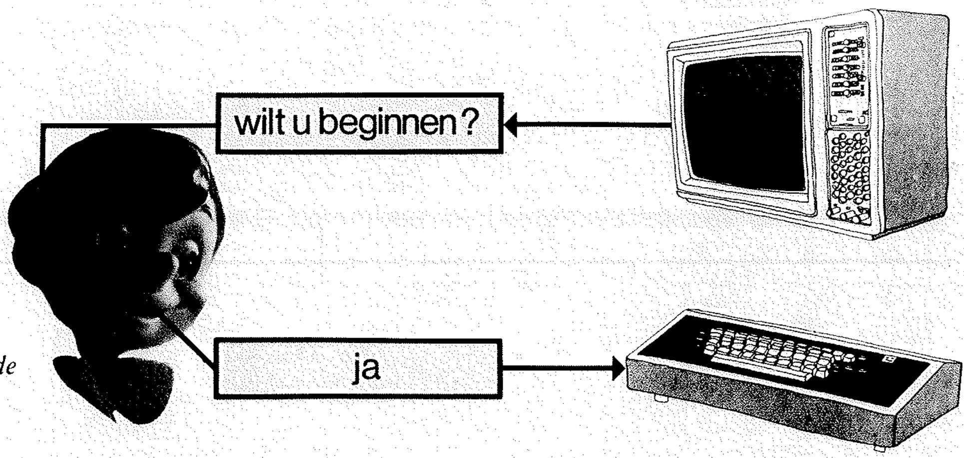
10 PRINT "TYPE EEN WAARDE IN"

Hierbij lieten we de tekst TYPE EEN WAARDE IN op het beeldscherm zetten. Deze tekst is volgens de hierboven gegeven definitie een string. We hebben zelf al aan onze computer meegedeeld dat we met een *string* te maken hebben. Deze is namelijk door ons tussen dubbele aanhalingstekens (") geplaatst. Op grond hiervan wist de computer dat hij met een string van gewone tekst te maken had. Een ander voorbeeld hadden we bij het INPUT statement:

10 INPUT "HOE OUD BENT U"; JR

Ook hier gebruikten we al een string, namelijk "HOE OUD BENT U". In dit geval is die string eveneens tussen dubbele aanhalingstekens geplaatst (in het Engels wordt het aanhalingsteken *quote* genoemd).
We zien uit deze voorbeelden dat de string wordt gebruikt in een soort gesprek tussen de computer en de bediener van de computer. In beide voorbeelden zet de computer een zin op het beeldscherm en de gebruiker moet als antwoord een getal intypen.

Het zou zeer wenselijk zijn wanneer de gebruiker ook gewoon een tekst zou mogen intypen, zoals bijvoorbeeld JA of NEE. Dan kan een veel eenvoudiger conversatie ontstaan (zie afbeelding 10.2). Wan-



Afb. 10.2. Conversatie met de computer

neer de computer op deze tekst moet kunnen reageren, dan moet hij allereerst kunnen herkennen dat we met tekst werken. Ten tweede moet er dan iets met die tekst worden gedaan.
In de vorige hoofdstukken hebben we gezien dat er variabelen zijn, waaraan we een bepaalde getalwaarde toekennen. Zo zijn er ook variabelen, waaraan we strings kunnen toekennen. We noemen deze variabelen *alfanumerieke variabelen* (zie ook hoofdstuk 2). De naam *stringvariabelen* wordt ook wel in plaats van alfanumerieke variabelen gebruikt.
Om deze alfanumerieke variabelen van de getalvariabelen te onderscheiden moet in BASIC zo'n variabele altijd met een *dollarteken* ($) worden geëindigd.

Legale alfanumerieke variabele namen zijn bijvoorbeeld:

A$
N$

Dit betekent dat de variabele A getalwaarden kan bevatten, maar geen string, terwijl de variabele A$ een string bevat en geen getalwaarde.

Voorbeeld

```
*LIST
10    LET A$="IK VINDT JE ZO AARDIG"
20    PRINT A$
*RUN
IK VINDT JE ZO AARDIG
*
```

We hebben nu gezien dat er alfanumerieke variabelen bestaan. Deze moeten met een $ worden afgesloten. De string moet tussen dubbele aanhalingstekens staan. Overigens: u ziet in bovenstaand voorbeeld dat door u ingetypte taalfouten (Ik vindt) keurig bewaard blijven. De computer corrigeert dus niet.

10.2. Stringoperaties

We kunnen nu de meeste operaties, die in de vorige hoofdstukken ter sprake zijn gekomen, ook op zo'n alfanumerieke variabele laten werken.

Voorbeeld

```
*LIST
10    LET A$="MOOI WEER"
20    PRINT A$
30    INPUT "OF NIET SOMS";B$
40    IF B$="JA" GOTO 80
50    FOR I=1 TO 4
60    PRINT A$;" TOCH WEL"
70    NEXT I
80    PRINT "BEDANKT"
90    END
*RUN
MOOI WEER
OF NIET SOMS
?NEE
MOOI WEER TOCH WEL
MOOI WEER TOCH WEL
MOOI WEER TOCH WEL
MOOI WEER TOCH WEL
BEDANKT
END PROGRAM
*RUN
MOOI WEER
OF NIET SOMS
?JA
BEDANKT
END PROGRAM
*
```

In dit programma kennen we op regel 10 aan de stringvariabele A$ de string MOOI WEER toe. Op regel 20 printen we deze uit. Zo'n samentrekking wordt in het Engels *concatenation* genoemd (in het Nederlands soms *concatenatie*).
Regel 30 print nu de vraag OF NIET SOMS? uit (weer een string) en vraagt als invoer om de stringvariabele B$. In regel 40 wordt bezien of het antwoord JA is. Als het antwoord JA is, printen we BEDANKT.
Wanneer het antwoord niet JA is, gaan we even pesten en laten via de regels 50 tot en met 70 viermaal de tekst MOOI WEER TOCH WEL printen. Het opvallende van regel 60 is dat we als het ware 2 strings tijdens het printen aaneenrijgen; de alfanumerieke variabele A$, die de string MOOI WEER bevat en de string TOCH WEL.

Een ander klein voorbeeld
We maken een subroutine, die nagaat of op een vraag het antwoord JA of NEE is gegeven. Als geen van beide antwoorden wordt ingetypt, wordt de vraag opnieuw herhaald.

Bij de invoer van een string met INPUT staat de Teleac/DAI personal computer toe dat deze string tussen dubbele aanhalingstekens wordt ingetypt. Alles wat tussen deze dubbele aanhalingstekens (") staat, wordt in de string opgenomen. Het is echter ook mogelijk de string zonder de dubbele aanhalingstekens in te voeren. Dit heeft wel consequenties. In de eerste plaats worden vooroplopende blanco's genegeerd en in de tweede plaats mag de string niet met een *cijfer*, maar moet met een *letter* beginnen.
De bedoelde subroutine staat op de regels 70 tot en met 150 van onderstaand voorbeeld:

```
*LIST
10    REM TEKSTDEMONSTRATIE
20    LET V$="MOOI WEER VANDAAG"
30    GOSUB 90
40    LET V$="GISTEREN OOK"
50    GOSUB 90
60    GOTO 160
70    REM DIT IS DE JA-NEE SUBROUTINE
80    REM AAN TE ROEPEN OP REGEL 90
90    PRINT V$
100   INPUT A$
110   IF A$="JA" GOTO 150
120   IF A$="NEE" GOTO 150
130   PRINT "HET ANTWOORD MOET JA OF NEE ZIJN"
140   GOTO 90
150   RETURN
160   END
*RUN
MOOI WEER VANDAAG
?"JAA"
HET ANTWOORD MOET JA OF NEE ZIJN
MOOI WEER VANDAAG
?"JA"
GISTEREN OOK
```

(vervolg programma)

```
?"NEEN"
HET ANTWOORD MOET JA OF NEE ZIJN
GISTEREN OOK
?NEEN
HET ANTWOORD MOET JA OF NEE ZIJN
GISTEREN OOK
?NEE
END PROGRAM
*
```

Hier zien we dan ook weer dat strings kunnen worden toegekend, ingevoerd, vergeleken en geprint.
Relationele operaties (vergelijkingen) op strings zijn toegestaan. Hiervoor gelden nog enige overwegingen.
Bij strings van dezelfde lengte wordt links beginnend de relationele operatie op de waarde van de tekens uitgevoerd. Er wordt uitgegaan van de waarde, die een teken in de ASCII-tabel heeft (zie afbeelding 10.3) Een kleinere waarde in de tabel geeft daarbij een kleinere waarde voor dat teken.

ASCII-code: $b_7 b_6 b_5 b_4 b_3 b_2 b_1$

b_4 b_3 b_2 b_1	b_7 → b_6 → b_5 →	0 0 0	0 0 1	0 1 0	0 1 1	1 0 0	1 0 1	1 1 0	1 1 1
0 0 0 0		NUL	DLE	SPACE	0	@	P	`	p
0 0 0 1		SOH	DC1	!	1	A	Q	a	q
0 0 1 0		STX	DC2	"	2	B	R	b	r
0 0 1 1		ETX	DC3	#	3	C	S	c	s
0 1 0 0		EOT	DC4	$	4	D	T	d	t
0 1 0 1		ENQ	NAK	%	5	E	U	e	u
0 1 1 0		ACK	SYN	&	6	F	V	f	v
0 1 1 1		BEL	ETB	'	7	G	W	g	w
1 0 0 0		BS	CAN	(	8	H	X	h	x
1 0 0 1		HT	EM	)	9	I	Y	i	y
1 0 1 0		LF	SUB	*	:	J	Z	j	z
1 0 1 1		VT	ESC	+	;	K	[	k	{
1 1 0 0		FF	FS	,	<	L	\	l	¦
1 1 0 1		CR	GS	-	=	M	]	m	}
1 1 1 0		SO	RS	.	>	N	∧	n	~
1 1 1 1		SI	US	/	?	O	—	o	DEL

Afb. 10.3. ASCII-tabel voor letters, cijfers en leestekens voor 7-bits woorden

Voorbeeld
A in de ASCII-tabel is 41H ofwel decimaal 65
B in de ASCII-tabel is 42H ofwel decimaal 66
Dus A < B

Dit klopt volgens het alfabet. Cijfers worden dus kleiner dan hoofdletters beschouwd. Hoofdletters zijn weer kleiner dan kleine letters.

```
*LIST
10    A$="ABC"
20    B$="BBC"
30    IF B$>A$ GOTO 50
40    GOTO 60
50    PRINT B$;" IS GROTER DAN ";A$
60    END
*RUN
BBC IS GROTER DAN ABC
END PROGRAM
*
```

Bij vergelijking van strings van verschillende lengtes wordt de kortste string met blanco's uitgebreid.

```
*LIST
10    A$="ABC"
20    B$="BB"
30    IF B$>A$ GOTO 50
40    GOTO 60
50    PRINT B$;" IS GROTER DAN ";A$
60    END
*RUN
BB IS GROTER DAN ABC
END PROGRAM
*
```

De logische en rekenkundige operaties mogen niet op strings worden toegepast. DE OPERATIE "+" (plus), toegepast op strings, heeft echter wel betekenis, nl. het aaneenrijgen van strings ofwel *concateneren.*

Voorbeeld

```
*LIST
10    PRINT "AAP "+"NOOT "+"MIES"
20    A$="AAP "
30    B$="NOOT"
40    C$=A$+B$
50    PRINT C$
*RUN
AAP NOOT MIES
AAP NOOT
```

Zie afbeelding 10.4 voor de operaties op strings.
In het DATA statements mogen strings en getallen door elkaar worden gebruikt.

LET A$ = "AAP"	Met let kan een string toegekend worden
PRINT A$	Print een string
INPUT A$	Voert een string in
DATA "AAP"	Zet een string in een datagebied
READ A$	Leest een string uit een datagebied
DIM A$ (10,10)	Definieert een array van strings
A$ + B$	Voegt 2 strings samen
A$ = B$	Vergelijkt strings
=, >, <, < >, < =, > =	Zijn toegestaan op strings

Afb. 10.4. Operaties op strings

Voorbeeld

```
*LIST
10     DATA "AAP ",25,30,"NOOT"
20     READ A$,A,B,B$
30     PRINT A$+B$
40     PRINT A+B
*RUN
AAP NOOT
 55
*
```

Het is zelfs mogelijk de string zonder dubbele aanhalingstekens in te voeren. De vooroplopende blanco's worden dan weer genegeerd zoals bij het INPUT statement gebeurt. Dit is uitsluitend ook bij het DATA statement mogelijk. De computer gaat dan na of er een variabele is met de naam zonder quotes ("). Wanneer die er niet is neemt hij aan dat er een *string* is bedoeld. Dit gebruik kan tot verwarring leiden wanneer het om een beter begrijpen van een programma gaat. **Daarom verdient het tussen quotes specificeren altijd de voorkeur!!!!!!!**

Dus ook het volgende voorbeeld is correct:

```
*LIST
10     DATA "AAP",NOOT,15
20     READ A$,B$,C
30     PRINT "GROTE ",A$
40     PRINT "GROTE ",B$
50     PRINT C
60     END
*RUN
GROTE        AAP
GROTE        NOOT
 15
END PROGRAM
*
```

We moeten er wel op letten dat we de informatie met de kenmerken uit het datagebied halen zoals we ze erin hebben gezet. Verwisseling van string en getallen mag derhalve niet optreden. Fout is dus bijvoorbeeld:

```
*LIST
10     DATA "AAP",NOOT,15
20     READ A$,B
30     PRINT A$
40     PRINT B
50     END
*RUN
TYPE MISMATCH IN LINE 20
OK
*
```

10.3. Stringarrays

Hoofdstuk 7 geeft aan dat voor getalwaarden *arrays* kunnen worden gevormd met het DIM statement. Daarin wijzen we met subscripts het door ons gewenste element aan. Met strings kan dat op dezelfde manier gebeuren. Zo zal DIM A$ (10, 10) een stringarray definiëren van 121 elementen, namelijk 11 rijen (0 tot en met 10 genummerd) en 11 kolommen.

Tot nu toe hebben we het nog niet over de toegestane lengte van de string gehad. In de Teleac/DAI personal computer mag de maximale lengte van een string 255 tekens zijn. Een spatie telt daarin ook mee als een teken. De minimale lengte is, voor een zogenaamde lege string, uiteraard 0.

DIM A$ (10, 10) definieert dus een stringarray, die maximaal 11 × 11 × 255 = 30855 tekens kan bevatten, als alle elementen de maximale lengte hebben. Terwijl DIM A (10, 10) ruimte reserveert voor 121 getallen, zal DIM A$ (10, 10) echter geen ruimte reserveren voor 30855 tekens. Er wordt slechts een begin-indicatie en een lengte-indicatie voor elk element van de array gereserveerd (zogenaamde *pointers)*. Deze pointers (aanwijzers) wijzen de beginpositie en de lengte in het computergeheugen van iedere string in de array aan. Daarmee kan de BASIC interpreter de string terugvinden en ophalen. De beginpositie en de lengte van een string definieert namelijk precies waar de string zich bevindt. Elk element kan een string van 255 karakters bevatten. Pas wanneer er met behulp van LET A$ (1, 1) = "AAP" een element wordt gevuld, zal de Teleac/DAI perso-

Voorbeeld

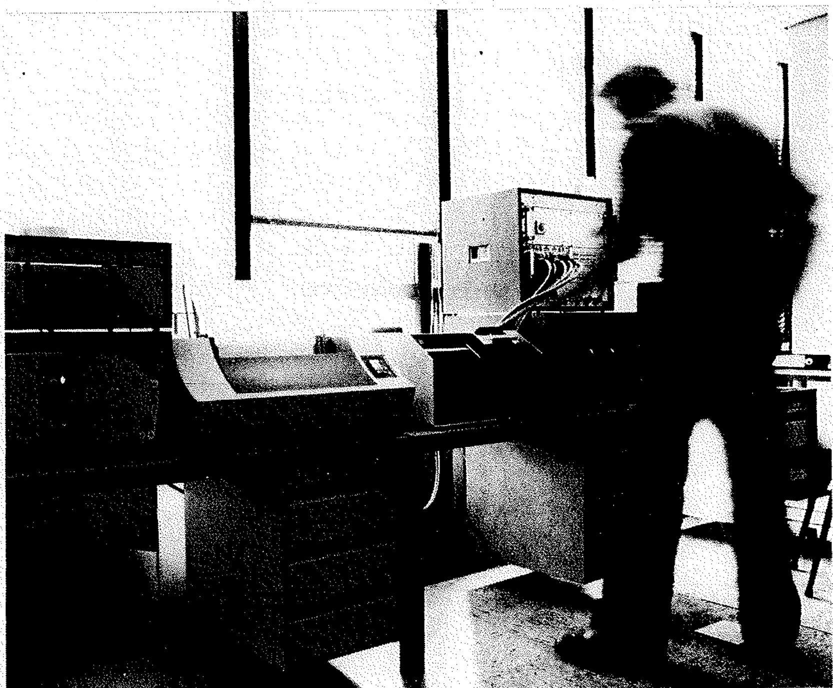
```
*LIST
10     DIM A$(2,3)
20     LET A$(0,0)="AAP"
30     LET A$(0,1)=" NOOT"
40     LET B$=A$(0,0)+A$(0,1)
50     PRINT B$
60     END
*RUN
AAP NOOT
END PROGRAM
*
```

nal computer de ruimte daarvoor ook echt gebruiken. De begin-indicatie wordt dan op de beginplaats van AAP en de lengte-indicatie op lengte 3 gezet (zie voorbeeld op blz. 176).
Er is een BASIC functie, genaamd FRE, die aangeeft hoeveel vrije geheugenruimte de programmeur nog ter beschikking staat.

Voorbeeld

```
*LIST
10    PRINT "NOG VRIJ";FRE;" BYTES"
*RUN
NOG VRIJ 3237 BYTES
*
```

Deze functie houdt daarbij in de Teleac/DAI personal computer geen rekening met de ruimte, die door strings in beslag wordt genomen. Dit komt omdat de lengte van een string niet op voorhand bekend is, zoals hierboven is weergegeven.
Veel personal computers kennen *geen* alfanumerieke variabelen in BASIC. Het komt ook dikwijls voor dat alfanumerieke variabelen *wel* zijn toegestaan, maar stringarray's weer *niet*. De Teleac/DAI personal computer geeft in de strings wel erg uitgebreide mogelijkheden t.a.v. de meeste andere personal computers.
Bij stringarray's hebben we de maximale lengte van een string bekeken. We zullen nu de maximale lengte van de naam van variabelen bezien.
De meeste BASIC versies staan een lengte van een of twee karakters toe (eventueel aangevuld met $). Daarom hebben we tot nu toe die beperking van één karakter opgelegd. De Teleac/DAI personal computer biedt evenwel de mogelijkheid de naam van een variabele uit veertien karakters te laten bestaan. Dit kan in hoge mate de leesbaarheid van programma's vergroten. Namen als RENTE, VERSCHIL,



Hoe snel men ook werkt, een miljoen acties per seconde kan de mens niet bijhouden

INGEVOERD%, FOUTMELDING$ en dergelijke zijn toegestaan. Het programma neemt dan wel meer geheugenruimte in beslag. De executie van het programma wordt ook enigszins vertraagd omdat de lange naam steeds herkend moet worden. Deze nadelen zijn, zoals reeds eerder is betoogd, ondergeschikt aan het voordeel van grotere leesbaarheid. In dit hoofdstuk beperken we ons nog tot namen van één karakter (plus eventueel de $ uiteraard). In de voorbeelden van hoofdstuk 11 zullen we ten behoeve van de leesbaarheid langere namen gebruiken.

10.4. Stringfuncties

In hoofdstuk 8 werd een aantal algemene BASIC functies besproken. Nu zullen we op de specifieke functies, die er voor strings bestaan, ingaan. Het argument voor zo'n functie moet altijd een string zijn. We kunnen de lengte van de string (LEN staat voor LENGTH = lengte) te weten komen door de functie LEN(X$) te gebruiken.

Voorbeeld

```
*LIST
10    REM DEMONSTRATIE VAN LEN
20    PRINT "TYPE EEN STRING IN"
30    INPUT I$
40    IF I$="END" GOTO 70
50    PRINT "LENGTE IS:";LEN(I$)
60    GOTO 20
70    END
*RUN
TYPE EEN STRING IN
?"WAT EEN SCHATJE"
LENGTE IS: 15
TYPE EEN STRING IN
?"     WAT EEN SCHATJE"
LENGTE IS: 20
TYPE EEN STRING IN
?WAT EEN SCHATJE
LENGTE IS: 15
TYPE EEN STRING IN
?     WAT EEN SCHATJE
LENGTE IS: 15
TYPE EEN STRING IN
?"1,2,3"
LENGTE IS: 5
TYPE EEN STRING IN
?1,2,3
TYPE MISMATCH RETYPE LINE
?1AAA
SYNTAX ERROR RETYPE LINE
?"1AAA"
LENGTE IS: 4
TYPE EEN STRING IN
?END
END PROGRAM
*
```

We zien nu nog eens de verschillende manieren van intypen van een string met INPUT geïllustreerd. Alles tussen dubbele aanhalingstekens wordt geaccepteerd. Zonder dubbele aanhalingstekens worden vooroplopende blanco's genegeerd. Verschillende variabelen op 1 regel moeten door komma's gescheiden worden.
Het komt nogal eens voor dat we in een nieuwe string een stuk tekst willen hebben, dat deel uitmaakt van een oude string. Daarvoor bestaan drie functies, die respectievelijk een linker, een rechter en een middendeel uit een string kunnen selecteren. Voor het selecteren van het linker deel beschikken we over de functie LEFT$.

Voorbeeld

```
*LIST
10    X$="BASIC"
20    PRINT LEFT$(X$,3)
*RUN
BAS
*
```

In regel 20 laten we van de string X$, die "BASIC" bevat, de drie tekens aan de linkerkant printen, dus "BAS".
De algemene vorm van de functie LEFT$ is: LEFT$ (X$, I) waarin X$ een alfanumerieke variabele is en I een geheel positief getal. I mag natuurlijk ook een expressie zijn, die een getal oplevert. (LEFT staat voor links). Het resultaat is dat we uit de stringvariabele X$ de linker I-tekens selecteren. I als lengte moet uiteraard groter of gelijk aan *nul* zijn. Als I groter is dan de lengte van X$ krijgen we een *foutmelding*. Als I negatief is, krijgen we ook een *foutmelding*. Wanneer I een floating point getal is, wordt de integerwaarde van I gebruikt. Het selecteren van *nul* tekens geeft een lege string.

Voorbeeld

```
*LIST
10    REM LEFT$ VOORBEELD
20    INPUT A$
30    FOR I=1 TO LEN(A$)+1
40    PRINT LEFT$(A$,I)
50    NEXT I
60    END
*RUN
?ZONNIG
Z
ZO
ZON
ZONN
ZONNI
ZONNIG
NUMBER OUT OF RANGE IN LINE 40
OK
*LIST
10    A$="ZONNIG"
20    PRINT LEFT$(A$,1.7)
*RUN
Z
*
```

We zien hier de werking van LEFT$. Ook is duidelijk te zien dat A$ en A als volstrekt verschillende variabelen worden gebruikt. Dit is hier alleen gedaan om het *verschil* te laten zien. U kunt dit beter in uw programma's *niet* navolgen. Er treedt namelijk snel begripsverwarring op. Het kiezen van (vrijwel) dezelfde naam voor variabelen met een verschillende betekenis druist ook volledig in tegen de programmeerregel van hoofdstuk 9.
De functie LEFT$ eindigt zijn naam ook met het dollarteken ($), omdat zijn resultaat opnieuw een string is. We zien hier ook dat LEFT$ een functie, die op twee argumenten werkt, namelijk de string waaruit moet worden geselecteerd en het aantal tekens dat moet worden geselecteerd.
Op gelijke wijze zal RIGHT$ (X$, I) uit de stringvariabele X$ de rechter I-tekens selecteren (RIGHT staat voor rechts).

Voorbeeld

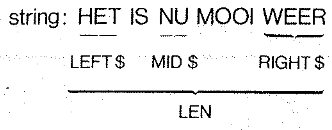
```
*LIST
10    A$="ZONNIG"
20    PRINT RIGHT$(A$,1.7)
*RUN
G
*
```

De functie MID$ kan vanaf een aangegeven positie uit een string n tekens selecteren. De algemene van MID$ is: MID$ (X$, I, J). Uit de string X$ zal vanaf positie I het aantal J-tekens worden geselecteerd (daarbij staat MID voor midden). De positietelling start daarbij vanaf Ø, dat wil zeggen het linker teken telt als het nulde teken, het op een na linkse als eerste, enz.

Voorbeeld

```
*LIST
10    REM VOORBEELDPROGRAMMA VAN MID$
20    INPUT A$
30    FOR N=0 TO LEN(A$)
40    PRINT MID$(A$,N,1),MID$(A$,N,2)
50    NEXT N
60    END
*RUN
?ZONNIG
Z         ZO
O         ON
N         NN
N         NI
I         IG
G
NUMBER OUT OF RANGE IN LINE 40
OK
*
```

We zien hierin ook de foutmelding NUMBER OUT OF RANGE, die signaleert dat we trachten buiten de string te adresseren. Deze functies worden nog eens samengevat in afbeelding 10.5.



LEN(X$)	**Geeft de lengte van de string**
LEFT$(X$,I)	**Geeft de linker I tekens van X$**
RIGHT$(X$,I)	**Geeft de rechter I tekens van X$**
MID$(X$,I)	**Geeft J tekens uit X$ beginnend op positie I**

Afb. 10.5. Selectie uit strings

10.5. Conversies

We hebben al gezien dat in onze Teleac/DAI personal computer de tekens via de ASCII-code worden weergegeven. Bij de vergelijking van strings wordt de binaire waarde uit de ASCII-tabel gebruikt. Hierbij vergelijken we de ASCII-waarden van elk character te beginnen bij positie Ø totdat er een verschil wordt gevonden ofwel beide strings leeg zijn. De programmeur kan deze ASCII-waarde ook via de functie ASC(X$) verkrijgen. (ASC staat daarin voor ASCII.) Deze functie zal de decimale waarde van de ASCII-code van het eerste teken in X$ als resultaat hebben.

Voorbeeld

```
*LIST
1Ø     REM VOORBEELD ASC
2Ø     PRINT "TYPE EEN TEKEN"
3Ø     INPUT T$
4Ø     IF T$="S" GOTO 7Ø
5Ø     PRINT ASC(T$)
6Ø     GOTO 2Ø
7Ø     END
*RUN
TYPE EEN TEKEN
?A
 65
TYPE EEN TEKEN
?ABCDEFG
 65
TYPE EEN TEKEN
?B
 66
TYPE EEN TEKEN
?O
 79
TYPE EEN TEKEN
?S
END PROGRAM
*
```

Het is een goede oefening met behulp van de in dit hoofdstuk gegeven ASCII-tabel de antwoorden nog eens te controleren.
Omgekeerd kunnen we ook een teken uit de ASCII-tabel laten genereren met behulp van de functie CHR$(I). (CHR staat voor character.) I moet daarbij tussen 0 en 256 liggen.

Voorbeeld

```
*LIST
10    REM VOORBEELD CHR$
20    INPUT "TYPE EEN GETAL";I
30    PRINT CHR$(I)
40    IF I<>1 GOTO 20
*RUN
TYPE EEN GETAL
?65
A
TYPE EEN GETAL
?66
B
TYPE EEN GETAL
?97
a
TYPE EEN GETAL
?270
NUMBER OUT OF RANGE IN LINE 30
OK
*
```

Zodra het ingetypte getal buiten het bereik valt (0-256) krijgen we een foutmelding te zien. Op deze wijze kunt u dus ook alle characters genereren, die in de ASCII-tabel zijn aangegeven. Sommige daarvan kunnen een bijzondere betekenis hebben, wat in de ASCII-tabel ook wordt aangegeven.
We kennen nu *getallen* en *strings*. Het is mogelijk om die in elkaar om te zetten. De functie STR$(X) geeft een string als resultaat, die de ASCII-weergave van de cijfers van het getal X is (STR staat in feite voor string).

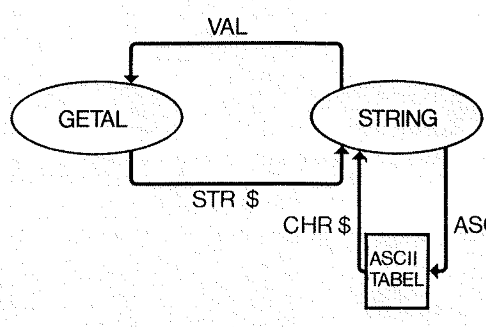
Voorbeeld

```
*LIST
10    REM VOORBEELD STR$
20    INPUT "TYPE EEN GETAL";G
30    PRINT G,STR$(G)
40    IF G<>1 GOTO 20
50    END
*RUN
TYPE EEN GETAL
?13
 13          13
TYPE EEN GETAL
?1.2E-10
 1.2E-10     1.2E-10
TYPE EEN GETAL
?-3.34
-3.34        -3.34
TYPE EEN GETAL
?1
 1           1
END PROGRAM
*
```

Het voordeel hiervan is dat we als het ware het getal in *tekstvorm* ter beschikking krijgen. Aangezien we gemakkelijk op strings kunnen opereren, kunnen we dat getal in een willekeurige door ons gewenste vorm brengen. Dit geldt met name als we getallen netjes onder elkaar of in tabellen naast elkaar willen afdrukken. In combinatie met de functies TAB, SPC en POS, die we verderop in dit hoofdstuk zullen behandelen, ontstaan dan fraaie opmaakmogelijkheden.
Van een tekst die uit cijfers bestaat kunnen we ook weer een getal maken, waarmee we kunnen rekenen. Hiervoor bestaat de functie VAL. (VAL staat voor VALUE = waarde). De functie VAL(X$) geeft als resultaat het getal, dat door de string X$ wordt weergegeven. Een schematisch beeld van de hier behandelde functies wordt in afbeelding 10.6 weergegeven.

Voorbeeld

```
 *LIST
10    REM VOORBEELD VAL
20    A$="1.2345"
30    G=VAL(A$)
40    PRINT G,G+G
50    PRINT A$,A$+A$
*RUN
 1.2345       2.469
1.2345       1.23451.2345
*
```



Afb. 10.6. String/getal omzettingen

We zien uit regels 40 en 50 duidelijk de verschillen tussen het getal en de string. Bij een getal is + de echte optelling, terwijl bij een string de + samenvoeging voorstelt. Wanneer bij VAL(A$) de variabele A$ geen getal voorstelt, krijgen we de foutmelding INVALID NUMBER.
De laatstgenoemde vier functies de zgn. *conversie-* of *vertalingsfuncties*. Zij zetten namelijk de verschillende vormen, waarin informatie aanwezig kan zijn, in de computer in elkaar om. Tekst kan daarbij naar getalwaarden worden omgezet en omgekeerd. Het is zo ook mogelijk een getal naar zijn waarde in de ASCII-code om te zetten en weer terug. Afbeelding 10.7 geeft hiervan nog eens een overzicht.

ASC(X$)	Geeft de ASCII waarde van het eerste teken uit X$
CHR$(I)	Geeft het ASCII teken met de waarde I
STR$(X)	Geeft de ASCII weergave van getalwaarde X
VAL(X$)	Geeft de getalwaarde van de string X$

Afb. 10.7. Conversiefuncties

10.6. Tekstopmaak

Nu worden nog drie functies behandeld, die te maken hebben met de *opmaak* van uit te voeren tekst en getallen.
Bij de PRINT commando hebben we gezien dat de uit te printen waarden door middel van een komma kunnen worden gescheiden. Ons scherm wordt dan als het ware ingedeeld in brede kolommen. Elk te printen getal of string begint op de kolomgrens. Wanneer we

de waarden door een puntkomma (;) scheiden, worden ze achter elkaar geprint. Getallen worden dan geprint met een blanco er tussen; strings worden zonder blanco's er tussen geprint. De blanco tussen getallen staan als het ware vooroplopend aan het getal zelf. Wanneer we een, of een; achter het PRINT statement laten staan, gaat het volgende PRINT statement weer op dezelfde regel verder op de plaats, waar het vorige PRINT statement voor het laatst geprint had.
In combinatie met het PRINT commando is het mogelijk om een waarde op een door ons te kiezen plaats te printen. Dit doen we met behulp van de TAB-functie. De TAB-functie is enigszins vergelijkbaar met de *tabulatorfunctie* op een schrijfmachine. TAB geeft de positie aan waarop geprint moet worden, gerekend van positie 0 als links zijnde. TAB kan alleen binnen een PRINT commando worden gebruikt (TAB staat voor tabulation = tabel).

Voorbeeld

```
*LIST
10    REM VOORBEELD VAN DE TAB$
20    FOR T=5 TO 10
30    PRINT TAB$(T);"MOOI";TAB$(25);T
40    NEXT T
50    END
*RUN
     MOOI                    5
      MOOI                   6
       MOOI                  7
        MOOI                 8
         MOOI                9
          MOOI               10
END PROGRAM
*P
```

Soms is het wenselijk niet de absolute positie, waar geprint moet worden, op te geven maar de positie relatief t.o.v. de laatste geprinte informatie. Dit kan met de functie SPC(J). (SPC staat voor *space* = ruimte.)
Deze functie print J-blanco's uit vanaf de plaats, waar het daaraan voorafgaande printen ophield, bij de *cursor*. De cursor is het teken op het beeldscherm, dat aangeeft waar het volgende character zal worden gezet.

Voorbeeld

```
*LIST
10    REM VOORBEELD VAN DE SPC$
20    PRINT "MOOI";SPC$(3);"WEER";SPC$(8);"VANDAAG"
30    PRINT SPC$(15);
40    PRINT "NOOT",12
*RUN
MOOI   WEER        VANDAAG
               NOOT      12
*
```

Deze 'opmaakfuncties' worden in afbeelding 10.8 nog eens samengevat. In TAB(X) en SPC(X) mag X uiteraard ook weer een *expressie* zijn die een getalswaarde oplevert.

PRINT A; B	**Print de waarden direct achter elkaar**
PRINT A; B	**Print de waarden op 14 koloms grenzen**
PRINT TAB (T); A	**Print A op positie T**
SPC (J)	**Geeft J spaties**

Afb. 10.8. Opmaakfuncties

Samenvatting hoofdstuk 10

In dit hoofdstuk is getoond hoe we tekst in BASIC kunnen bewerken. Een stuk tekst werd daarbij als een string aangeduid. Op zo'n string kunnen we veel operaties laten werken.
Daarnaast zijn er de speciale tekstbehandelingsfuncties LEFT$, RIGHT$ en MID$, die gedeelten uit een string selecteren. De conversiefuncties ASC, CHR$, STR$ en VAL zetten getallen en tekst in elkaar om. Voor tekstopmaak beschikken we tenslotte over de functies TAB en SPC. In TAB(X) en SPC(X) mag X uiteraard ook weer een expressie zijn, die een getalswaarde oplevert.

Multiple choice-vragen bij hoofdstuk 10

A. **a.** Invoer van strings dient altijd tussen dubbele aanhalingstekens te geschieden.
Strings mogen hoofdletters en kleine letters bevatten.

1 uitspraak a is goed en uitspraak b is goed
2 uitspraak a is fout en uitspraak b is goed
3 uitspraak a is goed en uitspraak b is fout
4 uitspraak a is fout en uitspraak b is fout

B. Uit hoeveel tekens mag een stringarray, gedefinieerd als DIM A$ (2, 3), in de Teleac/DAI-personal computer maximaal bestaan:

1 3060
2 12
3 6
4 1530

C. **a.** Alle relationele operaties zijn ook op strings toegestaan.
b. Elk teken uit een string-array kan door middel van subscripts worden aangegeven.

1 uitspraak a is goed en uitspraak b is goed
2 uitspraak a is fout en uitspraak b is goed
3 uitspraak a is goed en uitspraak b is fout
4 uitspraak a is fout en uitspraak b is fout

D. **a.** De functie ASC (X$) geeft de decimale waarden van de ASCII-code van alle tekens in X$.
b. De functie VAL (X$) geeft de waarde van de string van cijfers (X$) als getal weer.

1 uitspraak a is goed en uitspraak b is goed
2 uitspraak a is fout en uitspraak b is goed
3 uitspraak a is goed en uitspraak b is fout
4 uitspraak a is fout en uitspraak b is fout

E. **a.** De functie MID$ (X$, I, J) zou in principe voldoende zijn om elk deel uit een string X$ te selecteren.
b. Met STR$ (X) verkrijgen we het getal X als tekst-string.

1 uitspraak a is goed en uitspraak b is goed
2 uitspraak a is fout en uitspraak b is goed
3 uitspraak a is goed en uitspraak b is fout
4 uitspraak a is fout en uitspraak b is fout

F. **a.** De variabele A$ kan zowel getallen als characters bevatten.
b. Met het statement PRINT kunnen zowel getallen als strings worden geprint.

1 uitspraak a is goed en uitspraak b is goed
2 uitspraak a is fout en uitspraak b is goed
3 uitspraak a is goed en uitspraak b is fout
4 uitspraak a is fout en uitspraak b is fout

G. a. Bij de vergelijking van characters in twee strings wordt hun ASCII-waarde gebruikt.
b. Kleine letters zijn daarbij lager in waarde dan hoofdletters.

1 uitspraak a is goed en uitspraak b is goed
b uitspraak a is fout en uitspraak b is goed
3 uitspraak a is goed en uitspraak b is fout
4 uitspraak a is fout en uitspraak b is fout

H. a. Bij vergelijking van strings is de kortste string steeds kleiner in waarde dan een langere string.
b. De logische operaties en de rekenkundige operatie + (plus) mogen op strings worden toegepast.

1 uitspraak a is goed en uitspraak b is goed
2 uitspraak a is fout en uitspraak b is goed
3 uitspraak a is goed en uitspraak b is fout
4 uitspraak a is fout en uitspraak b is fout

I. a. Bij DIM A(2, 2) kan A maximaal uit negen getallen bestaan.
b. Bij DIM A$ (2, 2) kan A maximaal uit negen strings bestaan.

1 uitspraak a is goed en uitspraak b is goed
2 uitspraak a is fout en uitspraak b is goed
3 uitspraak a is goed en uitspraak b is fout
4 uitspraak a is fout en uitspraak b is fout

J. a. Een DATA statement mag zowel strings als numerieke variabelen bevatten.
b. Bij het uitzien via een READ statement mogen de verschillende soorten gegevens niet worden verwisseld.

1 uitspraak a is goed en uitspraak b is goed
2 uitspraak a is fout en uitspraak b is goed
3 uitspraak a is goed en uitspraak b is fout
4 uitspraak a is fout en uitspraak b is fout

Hoofdstuk 11

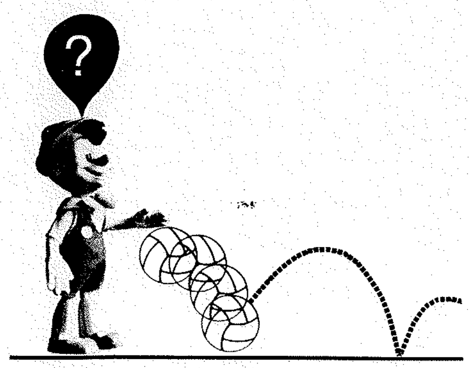
BASIC voorbeelden

In dit hoofdstuk zijn de regelnummers weer met nullen (ØØ) op 'computerwijze' afgedrukt.

In dit hoofdstuk zullen we een aantal voorbeelden behandelen van complete BASIC programma's. Bij deze voorbeelden gebruiken we terwille van de leesbaarheid van de programma's namen van meerdere characters. Die namen worden zo gekozen dat ze zo duidelijk mogelijk weergeven wat de betekenis van de variabele is. In de Teleac/DAI personal computer mogen de namen 14 characters lang zijn. In andere computers is dikwijls een beperking tot 1 of 2 characters aangebracht. Dan is deze vorm van leesbaarheidvergroting dus niet mogelijk. Soms mogen langere namen worden gebruikt, maar alleen de eerste twee characters bepalen de naam in het programma. De namen AAP, AAN en AARDE worden dan door het programma als hetzelfde beschouwd. Soms is het daarbij verboden in de namen keywords van BASIC te gebruiken.
Namen als LETTER, BALLET enz. zijn dan niet toegestaan, omdat ze het keyword LET bevatten.
Het is altijd mogelijk voor de oplossing van een probleem verschillende programma's te maken. De voorbeeldprogramma's geven dus zeker niet de enige manier om het gegeven probleem op te lossen.
Het is een zeer goede oefening om ook eens met een eigen programma te proberen het probleem op te lossen. Eventueel kan het aanbrengen van kleine wijzigingen in de hier weergegeven programma's als oefening gebruikt worden.

11.1. Stuitende bal

We nemen allereerst het voorbeeld van een stuitende bal.
Stel dat we die bal van een zekere hoogte laten vallen en we meten dan op hoe hoog de bal stuit. We nemen dan aan dat we iedere keer als de bal opnieuw stuit, dezelfde verhouding terug vinden. In werkelijkheid zal dit niet precies het geval zijn. Het gaat er hier echter niet om precies de werkelijkheid weer te geven, maar om te laten zien hoe we met BASIC kunnen rekenen. Vandaar dat we de gedane aanname acceptabel vinden.
De vraag is nu: hoe groot is de afstand, die de bal heeft afgelegd na een aantal keren stuiten? Zie afbeelding 11.1. Een bijbehorend programma wordt weergegeven in afbeelding 11.2.
In de regels 2Ø tot en met 4Ø vraagt het programma de vereiste gegevens op. Daarna print regel 5Ø een blanco regel. In regel 6Ø wordt de verhouding tussen stuithoogte en oorspronkelijke lengte berekend. Deze verhouding wordt verder op (regel 13Ø) steeds gebruikt om de nieuwe afgelegde afstand te berekenen. De eerste keer aan het



Afb. 11.1. Stuitende bal.

begin is de totale afstand Ø, terwijl er dan afgelegd zal worden OORSPRONG + STUIT. De regels 7Ø en 8Ø leggen dit vast.
In regel 9Ø printen we een kop in drie kolommen. De eigenlijke berekening per keer stuiten, vindt plaats in de regels 1ØØ tot en met 14Ø. In regel 11Ø tellen we de reeds afgelegde afstand (TOTAAL) op bij de deze keer af te leggen afstand (AFSTANDNU). Daarna printen we op regel 12Ø deze waarden uit. Op regel 13Ø rekenen we dan

```
*LIST
10    REM STUITENDE BAL PROGRAMMA
20    INPUT "OORSPRONKELIJKE HOOGTE";OORSPRONG
30    INPUT "HOOGTE NA 1 KEER STUITEN";STUIT
40    INPUT "HOE VAAK TE STUITEN";KEER
50    PRINT
60    LET VERH=STUIT/OORSPRONG
70    LET TOTAAL=0
80    LET AFSTANDNU=OORSPRONG+STUIT
90    PRINT "AANTAL","AFSTAND NU","TOTALE AFSTAND"
100   FOR K=1 TO KEER
110   LET TOTAAL=TOTAAL+AFSTANDNU
120   PRINT K,AFSTANDNU,TOTAAL
130   LET AFSTANDNU=AFSTANDNU*VERH
140   NEXT K
150   END
*RUN
OORSPRONKELIJKE HOOGTE
?100
HOOGTE NA 1 KEER STUITEN
?90
HOE VAAK TE STUITEN
?20
AANTAL      AFSTAND NU    TOTALE AFSTAND
 1           190           190
 2           171           361
 3           153.9         514.9
 4           138.51        653.41
 5           124.659       778.069
 6           112.193       890.262
 7           100.974       991.236
 8           90.8764       1082.11
 9           81.7887       1163.9
 10          73.6099       1237.51
 11          66.2489       1303.76
 12          59.624        1363.38
 13          53.6616       1417.05
 14          48.2954       1465.34
 15          43.4659       1508.81
 16          39.1193       1547.93
 17          35.2074       1583.13
 18          31.6866       1614.82
 19          28.518        1643.34
 20          25.6662       1669
END PROGRAM
```

Afb. 11.2

de afstand uit die bij de volgende keer stuiten zal worden afgelegd. De regels 1ØØ en 14Ø zorgen ervoor dat dit stuk het gewenste aantal keren wordt doorlopen.
Extra commentaarregels zijn in dit programma, behalve voor de kop, niet toegevoegd. De programma's zijn zo klein dat dit wel erg overdadig zou zijn. De INPUT en PRINT statements geven bovendien nog in leesbare vorm weer wat er gebeurt. Ditzelfde geldt voor de volgende twee programma's. Zodra de programma's iets groter worden zal er wel commentaar worden ingevoegd.
Een stroomdiagram verschaft in zo'n eenvoudig programma niet veel extra duidelijkheid. Dat is dus hier achterwege gelaten.

We kunnen het verloop van de stuitende bal ook grafisch weergeven. We geven daarbij in horizontale richting de afgelegde afstand bij elke keer stuiten weer. Verticaal geven we aan, het aantal keren dat er gestuit is.
Een programma dat dit voor ons doet is weergegeven in afbeelding 11.3.

We vragen eerst uiteraard weer de gegevens op en rekenen de verhouding en de eerst afgelegde afstand uit (regels 2Ø tot en met 7Ø). Ons scherm is maar 64 posities breed. We zullen 6Ø posities voor het printen gebruiken. De beginplaats wordt op positie 6Ø geprint. Daardoor introduceren we een verhouding tussen de echte afstand en wat we op het scherm weergeven. De verhouding of schaal berekenen we op regel 8Ø en printen deze uit in regel 9Ø. In de regels 11Ø tot en met 13Ø printen we de horizontale as, gebruikmakend van de TAB functie.
De regels 15Ø tot en met 18Ø geven de berekening van AFSTANDNU opnieuw weer. Met TAB vertalen we die weer in een te printen waarde. Let erop dat in regel 16Ø bij de te printen AFSTANDNU/SCHAAL de waarde Ø.5 is opgeteld. Dit is om de juiste afronding te krijgen. De TAB functie kapt zijn argumenten reeds af tot de vereiste integer waarde. Om de juiste waarde zo goed mogelijk te benaderen, zorgen we door die optelling voor de gewenste afronding. We zien dat we maar 6Ø posities gebruiken voor onze benadering. Vooral wanneer de te printen waarden klein zijn, zal een afwijking ontstaan van de door ons gewenste ideale lijn. We verspringen namelijk in stapjes van één positie. Alles wat kleiner is dan één positie kunnen we dus niet goed weergeven. De schaal geeft aan hoeveel dat in werkelijkheid is (in ons voorbeeld dus 3.16667). Wanneer de verschillen tussen opeenvolgende te printen waarden klein zijn, wordt de hier gebruikte weergave onnauwkeurig.

In hoofdstuk 18 zullen we zien dat de Teleac/DAI personal computer ook grafische modes kent, waarin we benaderingen beter kunnen doen plaatsvinden, namelijk tot aan 352 posities.
Het voordeel van de hier gebruikte methode is dat een aangesloten gewone printer het plaatje kan printen. Men moet zich evenwel bij die geprinte versie van de onnauwkeurigheid bewust zijn.
In de grafische modes kan het plaatje niet meer op een gewone printer worden afgedrukt, omdat deze zijn kop niet in voldoende kleine afstanden kan verplaatsen. Bovendien kan bij gebruik van kleur een printer de gebruikte kleuren niet weergeven.

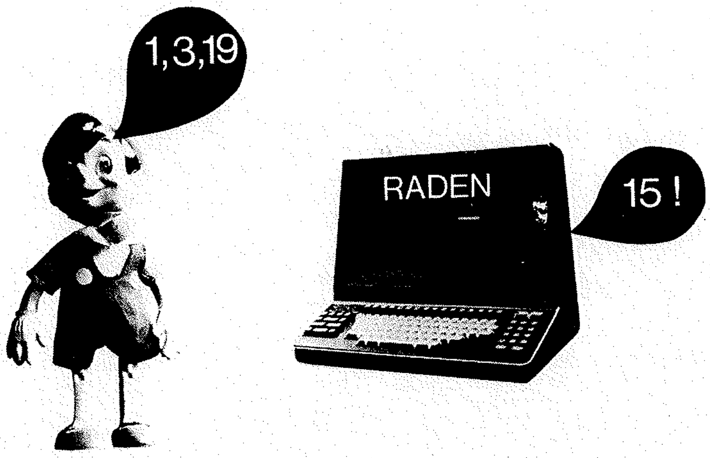
```
*LIST
10    REM STUITENDE BAL PRINTEN
20    INPUT "OORSPRONKELIJKE HOOGTE";OORSPRONG
30    INPUT "HOOGTE NA 1 KEER STUITEN";STUIT
40    INPUT "HOE VAAK TE STUITEN";KEER
50    PRINT
60    LET VERH=STUIT/OORSPRONG
70    LET AFSTANDNU=OORSPRONG+STUIT
80    LET SCHAAL=AFSTANDNU/60
90    PRINT "DE GETEKENDE SCHAAL IS 1 :";SCHAAL
100   PRINT
110   FOR J=0 TO 60
120   PRINT TAB(J);".";
130   NEXT J
140   PRINT
150   FOR K=1 TO KEER
160   PRINT ".";TAB(AFSTANDNU/SCHAAL+0.5);"*"
170   LET AFSTANDNU=AFSTANDNU*VERH
180   NEXT K
190   END
*RUN
OORSPRONKELIJKE HOOGTE
?100
HOOGTE NA 1 KEER STUITEN
?90
HOE VAAK TE STUITEN
?20


        DE GETEKENDE SCHAAL IS 1 : 3.16667
.............................................................
.                                                           *
.                                                     *
.                                                *
.                                           *
.                                      *
.                                  *
.                               *
.                            *
.                         *
.                      *
.                    *
.                  *
.                *
.              *
.             *
.           *
.          *
.         *
.        *
.       *
END PROGRAM
```

Afb. 11.3.

11.2. Getal raden

We gaan nu een zeer eenvoudig spelletje programmeren. U moet dan een getal in zo weinig mogelijk beurten raden (zie afbeelding 11.4). Een bijbehorend programma is afgedrukt in afbeelding 11.5.



Afb. 11.4. Getal raden.

In regel 2Ø vragen we het maximale getal op dat de computer in gedachten mag nemen. In regel 3Ø zetten we het om in een geheel getal vanaf 1 tot dat maximum. Regel 4Ø zet de telling van het aantal beurten op Ø. In regels 5Ø en 6Ø printen we uit dat de computer het getal in gedachten heeft genomen. Op regel 7Ø laten we raden. Op regel 9Ø en 1Ø kijken we of er goed geraden is. Op grond van de uitslag geven we dan de melding MIS. MIJN GETAL IS KLEINER of MIS. MIJN GETAL IS GROTER of GERADEN!!! IN ZOVEEL KEER.

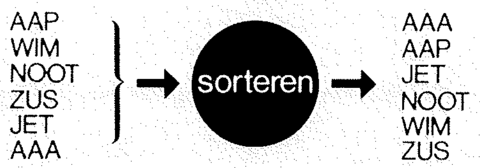
Bij verstandig raden heeft men voor een getal, dat binair in *n* bits kan worden weergegeven, nooit meer dan *n* beurten nodig. Het eerste raden slaat dan op het meest significante (hoogstwaardige) bit. Als het maximum getal, dat er geraden kan worden, 15 is, dan dus eerst 8 (het hoogstwaardige bit) raden. Komt daarna de boodschap MIJN GETAL IS GROTER, dan raden we 12 (8 + 4 ofwel het op één na hoogstwaardige bit erbij). Was de boodschap MIJN GETAL IS KLEINER dan raden we 4 (dat is na 8 het hoogstwaardige bit), enz. *Nota bene:* 15 is binair in 4 bits weergegeven: 1111. Men probeert dus de binaire codering af te lopen vanaf het meest significante bit naar het minst significante bit.

```
*LIST
10    REM GETAL RADEN PROGRAMMA
20    INPUT "WAT IS HET MAXIMALE GETAL";MAX
30    LET GETAL=1+INT(RND(MAX))
40    LET KEREN=0
50    PRINT "IK HEB EEN GEHEEL GETAL IN GEDACHTEN"
60    PRINT "TUSSEN 0 EN";MAX
70    INPUT "RAADT U MAAR";RAAD
80    LET KEREN=KEREN+1
90    IF GETAL=RAAD GOTO 150
100   IF GETAL>RAAD GOTO 130
110   PRINT "MIS,MIJN GETAL IS KLEINER"
120   GOTO 70
130   PRINT "MIS,MIJN GETAL IS GROTER"
140   GOTO 70
150   PRINT "GERADEN    GERADEN !!!"
160   PRINT "IN";KEREN;" KEER"
170   END
*RUN
WAT IS HET MAXIMALE GETAL
?15
IK HEB EEN GEHEEL GETAL IN GEDACHTEN
TUSSEN 0 EN 15
RAADT U MAAR
?8
MIS,MIJN GETAL IS KLEINER
RAADT U MAAR
?4
MIS,MIJN GETAL IS GROTER
RAADT U MAAR
?6
GERADEN    GERADEN !!!
IN 3 KEER
END PROGRAM
*RUN
WAT IS HET MAXIMALE GETAL
?15
IK HEB EEN GEHEEL GETAL IN GEDACHTEN
TUSSEN 0 EN 15
RAADT U MAAR
?8
MIS,MIJN GETAL IS KLEINER
RAADT U MAAR
?4
MIS,MIJN GETAL IS KLEINER
RAADT U MAAR
?2
MIS,MIJN GETAL IS KLEINER
RAADT U MAAR
?1
GERADEN    GERADEN !!!
IN 4 KEER
END PROGRAM
*
```

Afb. 11.5.

11.3. Sorteren

Ons volgend voorbeeld is het sorteren van een lijst met namen. Sorteren is iets dat bij het gebruik van de computer erg veel voorkomt. Vandaar dat er ook zeer veel methoden zijn ontwikkeld voor het sorteren. Iedere methode heeft zijn specifieke voor- en nadelen. Vooral wanneer al iets meer bekend is omtrent de te sorteren namen, kan een sorteermethode gekozen worden, die voor dat geval het meest aantrekkelijk is.
We zullen hier niet dieper ingaan op die verschillende methoden, maar er één als voorbeeld kiezen.
Onze methode staat in de literatuur bekend als *bubble sort.* Het mechanisme is hier op het volgende gebaseerd: Wanneer we via ieder opeenvolgend paar namen in een lijst zien dat ze in de juiste alfabetische volgorde staan, is de hele lijst op volgorde. Wanneer we echter een paar vinden, dat niet in de goede volgorde staat, verwisselen we de twee namen. We zetten dan tevens een geheugensteuntje op om ons er aan te herinneren dat we een paar hebben verwisseld. Wanneer we de hele lijst hebben doorlopen, kijken we naar ons geheugensteuntje. Hebben we dat opgezet, dan is er minstens één paar verwisseld en we beginnen weer opnieuw van voren af aan de lijst te doorlopen, terwijl we het geheugensteuntje terugnemen (zie voor één maal een lijst doorlopen afbeelding 11.6). We zien de zwart gemaakte naam naar voren schuiven. Afbeelding 11.7 geeft aan welke naam verplaatst is wanneer de volgende maal door de namen wordt gelopen. Zodra we nu aan het eind van de lijst ontdekken dat er geen geheugensteuntje opzij is gezet, heeft er geen verwisseling van namen meer plaats gehad. De lijst staat dus op volgorde.



Afb. 11.6. 'Bubble sort': AAA hobbelt naar voren in de eerste ronde.

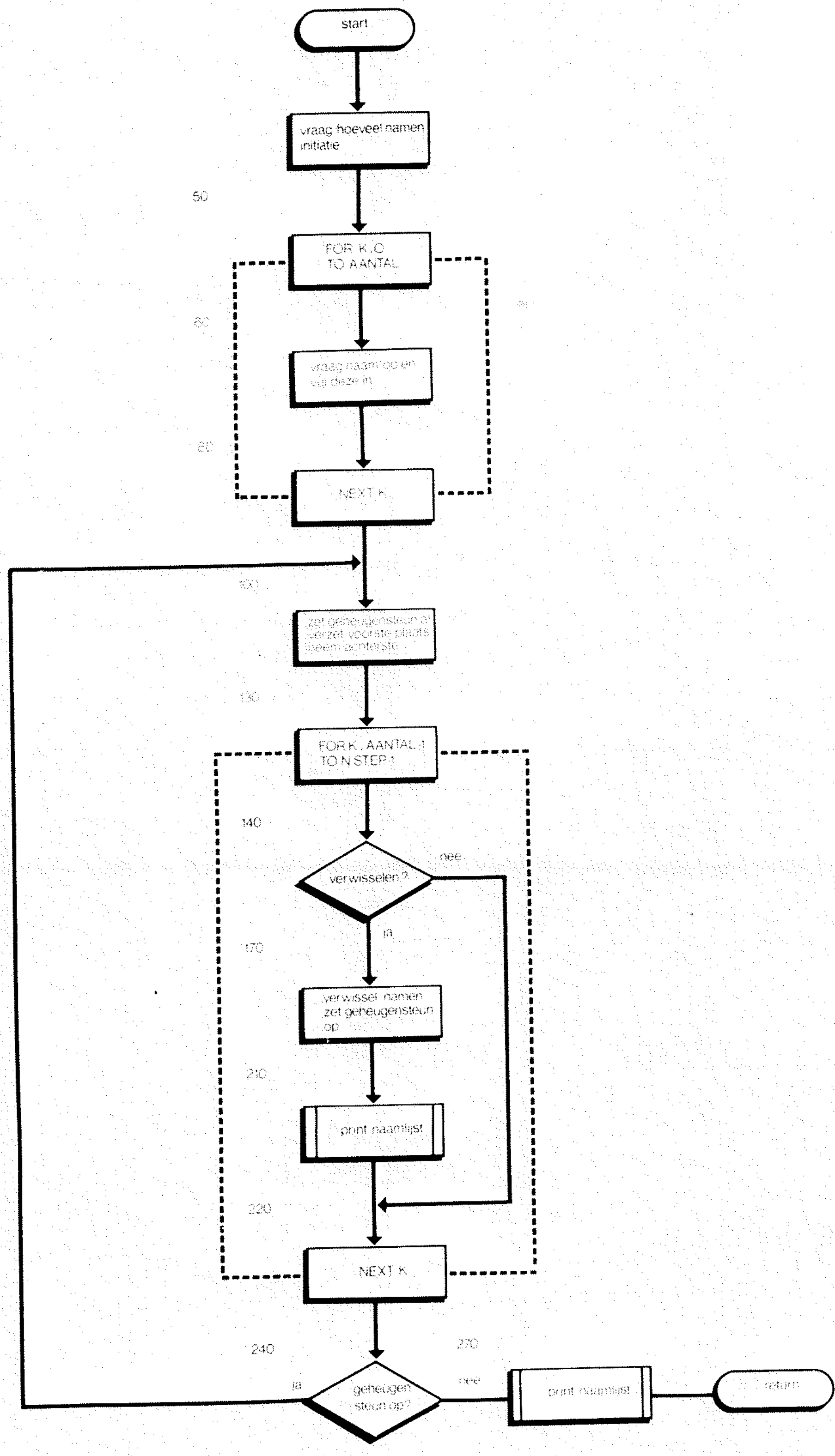
							verwisseling
beginwaarden:	AAP	WIM	NOOT	ZUS	JET	**AAA**	nee
na 1 slag:	AAP	WIM	NOOT	ZUS	**AAA**	JET	ja
na 2 slagen:	AAP	WIM	NOOT	**AAA**	ZUS	JET	ja
na 3 slagen:	AAP	WIM	**AAA**	NOOT	ZUS	JET	ja
na 4 slagen:	AAP	**AAA**	WIM	NOOT	ZUS	JET	ja
na 5 slagen:	**AAA**	AAP	WIM	NOOT	ZUS	JET	ja

AAA	**AAP**	JET	WIM	NOOT	ZUS
AAA	AAP	**JET**	NOOT	WIM	ZUS
AAA	AAP	JET	**NOOT**	WIM	ZUS
AAA	AAP	JET	NOOT	**WIM**	ZUS
AAA	AAP	JET	NOOT	WIM	**ZUS**

Afb. 11.7. De overige ronden in 'bubble sort'.

We verplaatsen de namen van achteren naar voren, zodat na één sorteerslag de juiste naam op de eerste plaats staat. Wanneer we nog iets slimmer werken, merken we op dat zodra we één maal door de lijst zijn gegaan, in ieder geval de juiste naam op de eerste plaats staat. Door voortdurende omwisseling is die naam namelijk net op die plaats gezet. De tweede maal hoeven we uiteraard niet meer naar die naam te kijken. We kunnen de nog door te lopen lijst met één naam bekorten. Zo komen we dus sneller klaar. Van dit mechanisme maken we in het voorbeeldprogramma dan ook gebruik. We printen ook de resultaten van de sorteerslagen uit, opdat het mechanisme nog eens goed kan worden gevolgd. Een stroomdiagram is gegeven in afbeelding 11.8. Een bijbehorend programma wordt gegeven door afbeelding 11.9.

In de regels 5Ø tot en met 8Ø vullen we de namenlijst LIJST op met de in te typen namen. In de regels 1ØØ tot en met 24Ø gaan we sorteren. Het één maal door de lijst gaan gebeurt op de regels 13Ø tot en met 22Ø. We zetten dan eerst de geheugensteun af (zie regel 11Ø). We bepalen nu tot hoe ver we nog moeten sorteren (regel 12Ø) en gaan dan per paar namen na of er moet worden omgewisseld (regels 17Ø, 18Ø en 19Ø). Daarvoor hebben we een hulpvariabele HULP$ nodig om even één naam bij het wisselen te bewaren. We zetten dan ook op regel 2Ø de geheugensteun op (1). Als we door de hele overgebleven lijst zijn gegaan, kijken we naar de geheugensteun (regel 24Ø). Staat die op (1), dan is er een omwisseling opgetreden en moeten we dus opnieuw door de lijst. Is dat niet het geval, dan printen we in regels 26Ø tot en met 28Ø de gesorteerde lijst uit. Voor het printen van de lijst gebruiken we een subroutine, die op regel 29Ø begint en op regel 33Ø eindigt. Dezelfde subroutine gebruiken we om iedere keer als er een verwisseling is opgetreden, de lijst uit te printen (regel 21Ø). Het verdient aanbeveling de RUN precies na te gaan. Daarmee wordt het algorithme verduidelijkt. Zoals al eerder is gesteld, hebben we hier slechts met een eenvoudig voorbeeld te maken. Als de lijst zeer lang wordt, kost het op deze manier sorteren veel tijd. In de Teleac/DAI personal computer is de maximum array-dimensie 255. Daar zit dus een duidelijke beperking in.
Voor grote naamlijsten treden daarom problemen op. In eerste instantie zijn die te ondervangen door de lijst in gedeelten te behandelen. Ieder stuk wordt afzonderlijk gesorteerd. Vervolgens komt er een programma in actie, dat van ieder stuk de eerste naam ophaalt en al die namen sorteert. Van het stuk, waaruit de nu eerste naam kwam, wordt vervolgens opnieuw een naam erbij gehaald. Daarna wordt er weer gesorteerd. Steeds wordt de eerste naam naar een resultaatlijst geschreven, totdat alle namen aan de beurt geweest zijn. Wanneer alle namen direct in het geheugen van de computer passen, kan dit nog redelijk snel plaatsvinden. Is de lijst echter zo groot dat de hele lijst niet meer in het computergeheugen past, dan komt de *cassetterecorder* er voor de opslag aan te pas. Voor een echt redelijk snel resultaat moet het systeem dan echter minstens over een *floppy disk* beschikken om herhaald spoelen op de recorder te voorkomen. Met ons voorbeeldprogramma hebben we dus echt het probleem van sorteren in grote lijsten niet opgelost. Daarvoor is veel moeilijker programmatuur (en apparatuur) nodig. Wel heeft u een inzicht kunnen krijgen hoe het sorteren zou kunnen gebeuren.



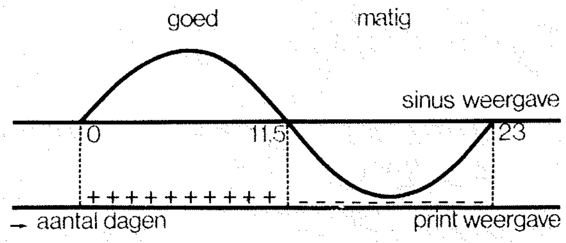
Afb. 11.8. Stroomdiagram sorteerprogramma

```
LIST
10   REM ZET EEN LIJST IN ALFABETISCHE VOLGORDE
20   REM EN PRINT HEM DAN UIT.
30   INPUT "HOEVEEL NAMEN WENST U";AANTAL
40   DIM LIJST$(AANTAL-1)
50   FOR K=0 TO AANTAL-1
60   INPUT "TYPE EEN NAAM IN";NAAM$
70   LET LIJST$(K)=NAAM$
80   NEXT K
90   LET N=0
100  PRINT "ER WORDT NU GESORTEERD"
110  LET GEHEUGENSTEUN=0
120  LET N=N+1
130  FOR K=AANTAL-1 TO N STEP -1
140  IF LIJST$(K-1)<=LIJST$(K) GOTO 220
150  REM VERWISSEL DE NAMEN
160  REM EN ZET DE GEHEUGENSTEUN OP.
170  LET HULP$=LIJST$(K-1)
180  LET LIJST$(K-1)=LIJST$(K)
190  LET LIJST$(K)=HULP$
200  LET GEHEUGENSTEUN=1
210  GOSUB 290
220  NEXT K
230  REM BEKIJK NU DE GEHEUGENSTEUN.
240  IF GEHEUGENSTEUN=1 GOTO 100
250  REM DE LIJST IS NU GESORTEERD.
260  PRINT "DE GESORTEERDE NAMENLIJST IS:"
270  GOSUB 290
280  GOTO 340
290  FOR J=0 TO AANTAL-1
300  PRINT LIJST$(J);" ";
310  NEXT J
320  PRINT
330  RETURN
340  END
*
*RUN
HOEVEEL NAMEN WENST U
?6
TYPE EEN NAAM IN
?"AAP"
TYPE EEN NAAM IN
?"WIM"
TYPE EEN NAAM IN
?"NOOT"
TYPE EEN NAAM IN
?"ZUS"
TYPE EEN NAAM IN
?"JET"
TYPE EEN NAAM IN
?"AAA"
ER WORDT NU GESORTEERD
AAP WIM NOOT ZUS AAA JET
AAP WIM NOOT AAA ZUS JET
AAP WIM AAA NOOT ZUS JET
AAP AAA WIM NOOT ZUS JET
AAA AAP WIM NOOT ZUS JET
ER WORDT NU GESORTEERD
AAA AAP WIM NOOT JET ZUS
AAA AAP WIM JET NOOT ZUS
AAA AAP JET WIM NOOT ZUS
ER WORDT NU GESORTEERD
AAA AAP JET NOOT WIM ZUS
ER WORDT NU GESORTEERD
DE GESORTEERDE NAMENLIJST IS:
AAA AAP JET NOOT WIM ZUS
END PROGRAM
*RUN
HOEVEEL NAMEN WENST U
?4
TYPE EEN NAAM IN
?A
TYPE EEN NAAM IN
?B
TYPE EEN NAAM IN
?C
TYPE EEN NAAM IN
?D
ER WORDT NU GESORTEERD
DE GESORTEERDE NAMENLIJST IS:
A B C D
END PROGRAM
*RUN
HOEVEEL NAMEN WENST U
?4
TYPE EEN NAAM IN
?D
TYPE EEN NAAM IN
?C
TYPE EEN NAAM IN
?B
TYPE EEN NAAM IN
?A
ER WORDT NU GESORTEERD
D C A B
D A C B
A D C B
ER WORDT NU GESORTEERD
A D B C
A B D C
ER WORDT NU GESORTEERD
A B C D
ER WORDT NU GESORTEERD
DE GESORTEERDE NAMENLIJST IS:
A B C D
END PROGRAM
*
```

Afb. 11.9.

11.4. Bioritme

Ons volgend voorbeeld behandelt het *bioritme*. Door sommige onderzoekers wordt beweerd dat de mens drie bioritmes kent. Een voor het lichaam, een voor het gevoel en een voor de intelligentie of geestelijke activiteiten. Vanaf de geboorte treden er volgens die verschillende ritmes fluctuaties op. Ieder ritme verloopt in een sinusvorming patroon (zie afbeelding 11.10).



Afb. 11.10. Lichaamsbioritme.

De mens is op de top van de sinus het best toegerust voor de bij dat ritme behorende eigenschap. In de nuldoorgangen, van boven naar beneden en omgekeerd, treedt er een kritieke situatie op. Dus als het lichaamsritme door een overgang gaat, is men niet tot optimale lichamelijke prestatie in staat. De kans op ziekte en dergelijke is dan groot.
De sinus begint met een positieve helft. Daarin zijn de perspectieven goed. Daarna volgt een negatieve helft, waarin de perspectieven matig zijn. De bewering luidt dat het lichaamsritme een cyclus van 23 dagen kent, het gevoel een cyclus van 28 dagen en het geestelijke vermogen een cyclus van 31 dagen. Kent men iemands geboortedag, dan is men in staat om op elke dag uit te rekenen hoe de de situatie is. Ons programma moet in staat zijn een overzicht voor één maand te geven. We drukken een voorbeeld van het gewenste overzicht af, waarbij in de rechterzijde van de sinus de perspectieven goed zijn, links is dat matig. Zie afbeelding 11.11.
Vervolgens geven we een bijbehorend BASIC programma weer in afbeelding 11.12. In dit geval zullen we niet de uitleg per regel geven. Het programma is van voldoende commentaar voorzien. Wel plaatsen we enige kanttekeningen.

In het begin van het programma worden de noodzakelijke gegevens opgevraagd. Er is een subroutine, die het aantal dagen in deze eeuw tot aan een gewenste dag berekent (regels 69Ø tot en met 78Ø). Daarbij wordt onderkend dat de maand februari in een schrikkeljaar 29 in plaats van 28 dagen telt. In de eeuwjaren, zoals 19ØØ, treedt geen schrikkeldag op. Het aantal dagen dat berekend moet worden, is samengesteld uit het aantal verlopen jaren maal 365 dagen (J * 365); het aantal verlopen schrikkeljaren (INT (J/4)) en het aantal verlopen dagen in het berekeningsjaar (A).
Bij het uitprinten van het overzicht worden na de kop drie sinusvormen geproduceerd. Als voorbeeld kijken we naar het lichaamsritme, regels 56Ø en 57Ø. Eerst wordt uitgerekend hoeveel dagen we al op weg zijn in de huidige sinus, namelijk FRAC (AANTAL/23).
Voor een mooi plaatje wordt deze waarde in radialen omgezet:
LICH = 2 * PI * FRAC(AANTAL/23).

```
BIORITME VAN VOORBEELD
GEBOORTEDATUM  21 3 46
VOOR JULI 79
GELEEFD BIJ HET OVERZICHTSBEGIN 12155 DAGEN

DAG       LICHAAM              GEVOEL               GEEST

 1            +                         +                  +
 2          +                            +                   +
 3       +                                +                   +
 4     +                                   +                  +
 5    +                                    +                   +
 6   +                                     +                   +
 7  +                                     +                    +
 8  +                                    +                    +
 9   +                                  +                    +
 1Ø   +                               +                     +
 11     +                           +                     +
 12        +                      +                      +
 13          +                  +                      +
 14            +              +                      +
 15               +         +                      +
 16                 +      +                      +
 17                  +    +                     +
 18                   +  +                     +
 19                   +  +                    +
 2Ø                  +   +                   +
 21                 +     +                  +
 22                +       +                 +
 23              +          +                 +
 24           +               +               +
 25         +                   +              +
 26      +                        +              +
 27    +                            +             +
 28   +                               +             +
 29  +                                  +             +
 3Ø +                                    +              +
 31 +                                     +               +

WEES VOORZICHTIG BIJ NULDOORGANGEN !!!!!
```

Afb. 11.11.

Als de fractie van AANTAL/23 van Ø tot 1 komt, moeten de radialen van Ø tot 2 * PI lopen. De sinusvorm wordt met behulp van TAB's, gevolgd door het teken + uitgeprint. De hoogte van een sinustop wordt als 9 gekozen, zodat drie sinussen goed op het scherm passen. Ten slotte moeten de sinussen naast elkaar worden geprint. Daarna krijgt ieder een extra verplaatsing op de regel. Voor het lichaamsritme is daarvoor 13.5 gekozen, zodat het nulpunt onder de H van LICHAAM uit de kop valt. De regel:

```
PRINT TAB (13.5 + 9 * SIN(LICH)) ; "+";
```

zorg nu voor het printen zelf.

Het construeren van kop en opschrift is een kwestie van nauwkeurig tellen, wat heel goed op ruitjespapier kan gebeuren. De regels 55Ø, 57Ø en 59Ø werden met een; afgesloten, zodat in dat geval niet op een nieuwe regel wordt geprint, maar we op de bestaande regel blijven. Dus na regel 61Ø zonder ; gaan we naar een nieuwe regel.
Het is een nuttige oefening een groter programma zoals dit eens in detail na te gaan. Daarbij verdient het als oefening aanbeveling kleine aanpassingen en verbeteringen uit te proberen. Ook hier is het programma weer zo eenvoudig van structuur, dat een stroomdiagram niet tot extra duidelijkheid aanleiding geeft. Het programma loopt recht toe recht aan zonder veel beslissingen. Het stroomdiagram zou de structuur verduidelijken, maar die is in dit geval reeds zeer eenvoudig.

Afb. 11.12

```
*LIST
1Ø    REM BIORITME PROGRAMMA
2Ø    DIM MAANDNAMEN$(12),DAGENTOTMAAND(12)
3Ø    REM GEGEVENS OPVRAGEN
4Ø    INPUT "TYPE UW NAAM":NAAM$
5Ø    PRINT "TYPE UW GEBOORTEDATUM IN DE VOLGORDE:"
6Ø    PRINT "DAG,MAAND,JAAR BYV.21,3,46"
7Ø    INPUT GDAG,GMAAND,GJAAR
8Ø    PRINT "TYPE MAAND EN JAAR VAN HET OVERZICHT"
9Ø    INPUT OMAAND,OJAAR
1ØØ   REM LEES DE MATRIX DAGENTOTMAAND IN
11Ø   FOR K=1 TO 12
12Ø   READ DAGENTOTMAAND(K)
13Ø   NEXT K
14Ø   REM BEREKEN NU HET AANTAL DAGEN VANAF HET BEGIN VAN
15Ø   REM DE EEUW TOT AAN DE GEBOORTEDAG.
16Ø   LET D=GDAG
17Ø   LET M=GMAAND
18Ø   LET J=GJAAR
19Ø   REM ROEP DE BEREKENINSSUBROUTINE AAN DIE ALS INVOER
2ØØ   REM D,J EN M VEREIST TERWIJL A HET RESULTAAT IS.
21Ø   GOSUB 69Ø
22Ø   LET GAANTAL=A
23Ø   REM HETZELFDE VOOR HET BEGIN VAN DE OVERZICHTSMAAND
24Ø   LET D=1
25Ø   LET M=OMAAND
26Ø   LET J=OJAAR
27Ø   GOSUB 69Ø
28Ø   LET OAANTAL=A
29Ø   LET AANTAL=OAANTAL-GAANTAL
3ØØ   REM ALS AANTAL<Ø KUNNEN WE GEEN OVERZICHT PRODUCEREN
31Ø   IF AANTAL<Ø GOTO 5Ø
32Ø   REM BEREKEN HET AANTAL DAGEN IN DE OVERZICHTSMAAND
33Ø   LET DAGEN=31
```

```
340 ON OMAAND GOTO 4ØØ,37Ø,4ØØ,35Ø,4ØØ,35Ø,4ØØ,4ØØ,35Ø,4ØØ,35Ø,4ØØ
35Ø LET DAGEN=3Ø
36Ø GOTO 4ØØ
37Ø LET DAGEN=28
38Ø REM CORRECTIE VOOR FEBRUARI IN EEN SCHRIKKELJAAR
39Ø IF (FRAC(OJAAR/4)=Ø) AND (OJAAR<>Ø) THEN DAGEN=DAGEN+1
4ØØ REM NU ZOEKEN WE DE MAANDNAAM OP
41Ø FOR L=1 TO 12
42Ø READ MAANDNAMEN$(L)
43Ø NEXT L
44Ø REM NU PRODUCEREN WE DE OVERZICHTSKOP
45Ø PRINT
46Ø PRINT "BIORITME VAN ";NAAM$
47Ø PRINT "GEBOORTEDATUM ";GDAG;GMAAND;GJAAR
48Ø PRINT "VOOR ";MAANDNAMEN$(OMAAND);OJAAR
49Ø PRINT "GELEEFD BIJ HET OVERZICHTSBEGIN";AANTAL;" DAGEN"
5ØØ PRINT
51Ø PRINT "DAG";SPC$(7);"LICHAAM";SPC$(14);"GEVOEL";SPC$(15);"GEEST"
52Ø PRINT
53Ø REM NU HET OVERZICHT PRINTEN
54Ø FOR M=1 TO DAGEN
55Ø PRINT M;
56Ø LET LICH=2*PI*FRAC(AANTAL/23)
57Ø PRINT TAB(13.5+9*SIN(LICH));"+";
58Ø LET GEV=2*PI*FRAC(AANTAL/28)
59Ø PRINT TAB(33.5+9*SIN(GEV));"+";
6ØØ LET GST=2*PI*FRAC(AANTAL/31)
61Ø PRINT TAB(53.5+9*SIN(GST));"+"
62Ø AANTAL=AANTAL+1
63Ø NEXT M
64Ø REM NU HET ONDERSCHRIFT PRINTEN
65Ø PRINT
66Ø PRINT "WEES VOORZICHTIG BIJ NULDOORGANGEN !!!!!"
67Ø REM WE ZIJN NU KLAAR
68Ø GOTO 86Ø
69Ø REM BEREKENINGSSUBROUTINE VOOR HET AANTAL DAGEN
7ØØ REM VANAF HET BEGIN VAN DE EEUW.
71Ø REM INPUT D M J VOOR DAG MAAND EN JAAR
72Ø REM OUTPUT A VOOR HET AANTAL DAGEN
73Ø REM DE MATRIX DAGENTOTMAAND WORDT HIER GEBRUIKT
74Ø LET A=DAGENTOTMAAND(M)+D-1
75Ø REM SCHRIKKELJAAR CORRECTIE
76Ø IF (M>2) AND (FRAC(J/4)=Ø) AND (J<>Ø) THEN A=A+1
77Ø LET A=A+(J*365)+INT(J/4)
78Ø RETURN
79Ø REM NU DE DATA STATEMENTS
8ØØ REM EERST VOOR DAGENTOTMAAND
81Ø DATA Ø,31,59,9Ø,12Ø,151,181,212,243,273,3Ø4,334
82Ø REM DAN VOOR MAANDNAMEN$
83Ø DATA "JANUARI","FEBRUARI","MAART","APRIL","MEI","JUNI"
84Ø DATA "JULI","AUGUSTUS","SEPTEMBER","OKTOBER","NOVEMBER"
85Ø DATA "DECEMBER"
86Ø END
*
```

Vervolg afb. 11.12.

```
RUN
TYPE UW NAAM
?BABY
TYPE UW GEBOORTEDATUM IN DE VOLGORDE:
DAG,MAAND,JAAR BYV.21,3,46
?1,7,79
TYPE MAAND EN JAAR VAN HET OVERZICHT
?7,79
BIORITME VAN BABY
GEBOORTEDATUM  1 7 79
VOOR JULI 79
GELEEFD BIJ HET OVERZICHTSBEGIN Ø DAGEN


DAG       LICHAAM              GEVOEL                GEEST

  1           +                   +                   +
  2             +                   +                   +
  3                +                  +                   +
  4                  +                  +                   +
  5                   +                  +                   +
  6                    +                  +                   +
  7                    +                   +                  +
  8                   +                    +                   +
  9                  +                     +                   +
 1Ø                 +                     +                    +
 11               +                      +                    +
 12           +                         +                    +
 13         +                         +                     +
 14      +                          +                     +
 15    +                          +                      +
 16   +                         +                      +
 17  +                        +                      +
 18 +                       +                      +
 19 +                      +                      +
 2Ø  +                    +                     +
 21   +                  +                     +
 22     +                +                    +
 23        +             +                   +
 24          +            +                  +
 25            +           +                 +
 26               +         +                 +
 27                  +        +               +
 28                   +         +              +
 29                    +          +              +
 3Ø                    +            +             +
 31                   +               +             +
```

Vervolg afb. 11.12.

```
WEES VOORZICHTIG BIJ NULDOORGANGEN !!!!!
END PROGRAM
```

Het kan als documentatie zeer nuttig zijn de betekenis van de gebruikte variabelen in het begin van het programma via REM statements weer te geven. In bovenstaand geval zou de tekst kunnen luiden:

REM NAAM$:	de ingetypte naam
REM GDAG:	de geboortedag
REM GMAAND:	de geboortemaand
REM GJAAR:	het geboortejaar
REM OMAAND:	overzicht maand
REM OJAAR:	overzicht jaar
REM D, M, J:	hulp input variabelen van de subroutine
REM A:	hulp output variabele van de subroutine
REM GAANTAL:	aantal dagen van deze eeuw op de geboortedag
REM OAANTAL:	aantal dagen van deze eeuw bij het overzichtsbeging
REM AANTAL:	aantal geleefde dagen
REM K, L:	hulp telvariabelen
REM DAGEN TOT MAAND:	aantal dagen in een jaar tot het begin van de maand
REM DAGEN:	aantal dagen van de overzichtsmaand
REM MAANDNAMEN$:	hulp-array met de namen van iedere maand
REM LICH:	lichamelijke gesteldheid
REM GEV:	gevoelsgesteldheid
REM GST:	geestelijke gesteldheid

Hier zijn de variabelen in gebruiksvolgorde van het programma opgesomd. Het kan evenzeer zinvol zijn de opsomming alfabetisch te laten verlopen.
Het programma kent weinig beveiligingen tegen foutief ingetypte gegevens. Wanneer de gebruiker bijvoorbeeld vraagt om een overzicht van maand 15 in 1979, loopt het programma in regel 740 fout. Dit is afgedrukt in afbeelding 11.13.

```
 RUN
TYPE UW NAAM
?FOUTJE
TYPE UW GEBOORTEDATUM IN DE VOLGORDE:
DAG,MAAND,JAAR BYV.21,3,46
?3,4,5
TYPE MAAND EN JAAR VAN HET OVERZICHT
?15 ,79
SUBSCRIPT ERROR IN LINE 740
OK
```

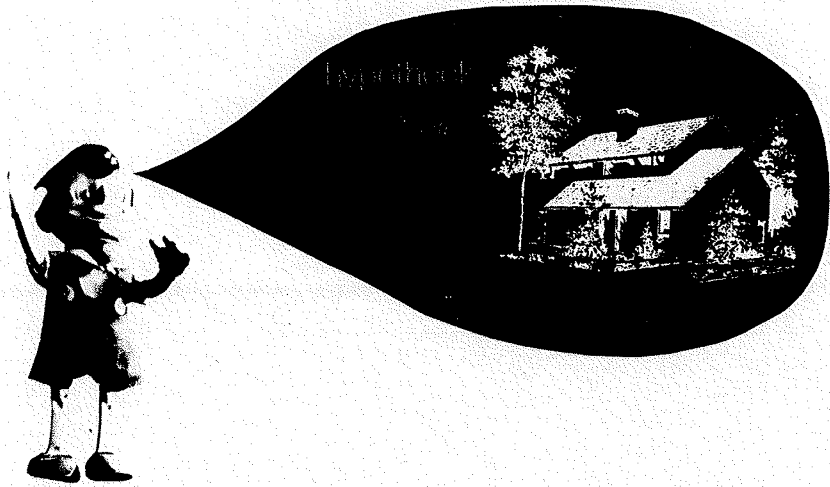
Afb. 11.13.

Programma's, die ook tegen dit soort fouten beveiligd moeten zijn, moeten ieder antwoord van de gebruiker eerst uitgebreid nagaan. Daarbij moet worden bezien of het antwoord wel juist kan zijn. Het

kan voorkomen dat de gebruiker wordt gevraagd om opnieuw in te typen. Het is een nuttige oefening dit soort beveiligingen in bovenstaand programma aan te brengen.
Wanneer het programma het resultaat op het beeldscherm zet in plaats van op een printer, is het scherm te klein (24 regels) om alle regels te bevatten. Na een aantal geprinte regels zou een STOP statement moeten worden ingebouwd om de eerste regels te kunnen bekijken. Het programma stopt dan en kan door de gebruiker via CONT weer worden hervat om de rest van de regels te bezien, zoals getoond in hoofdstuk 6.
Bij de Teleac/DAI personal computer kan de output ook tijdelijk worden gestopt en weer worden hervat met behulp van de spatiebalk. Sommige computers bezitten een aparte toets hiervoor.

11.5. Lineaire hypotheek

We gaan nu de kosten, verbonden aan een lineaire hypotheek berekenen, zoals weergegeven in afbeelding 11.14. Bij een lineaire hypotheek is de aflossing een constant bedrag. Het hypotheekbedrag wordt in de looptijd van de hypotheek geheel afgelost.



Afb. 11.14. Hypotheekberekening.

We veronderstellen dat de te betalen rente gelijk is aan het rentepercentage maal de resterende schuld. Per bank kunnen daarin nog verschillen optreden.
Zo kunnen ook in de renteberekening bij verschillende banken verschillen optreden. De ene bank rekent 12% per jaar. Dan rekent ze de totale rente per jaar uit en deelt dat door 12. Zo ontstaat de rente per maand. Maar anderen nemen 1% per maand. Dit is echter 12.68% per jaar. Wij gaan van de laatste berekeningswijze uit zoals uit het programma zal blijken, namelijk −1+(1.Ø1) ↑ 12%.
Bij de belastingberekening gaan we uit van de tabel van 1979. Daarbij schatten we de economische waarde van het huis op Ø.7 × de koopwaarde.
Er wordt ook rekening gehouden met het feit dat meestal een extra levensverzekering moet worden afgesloten. Tenslotte nemen we nog

de vaste kosten van het huis mee. Ook wordt de inflatie meegerekend. Dit laatste is uiteraard zeer speculatief.
Wij veronderstellen een constant inflatiepercentage gedurende de looptijd van de hypotheek. Daarbij veronderstellen we dat de schuld, het te betalen hypotheekbedrag en de levensverzekeringspremie aan inflatie onderhevig is. Bij de belastbare som, de huurwaarde en de belastingschijven veronderstellen we geen inflatie. Met andere woorden: gedurende de looptijd van de hypotheek blijven die geldbedragen evenveel waard als ze nu zijn. Dus: geen reële salarisverhoging en geen extra verzwarende belastingmaatregelen. We houden ook een constant blijvend rentepercentage aan.
In het voorbeeldprogramma houden we geen rekening met subsidie en extra aftrekbare kosten in het eerste jaar. Deze verfijningen moet u zelf maar eens als oefening aanbrengen. Ook als u van andere veronderstellingen als hierboven weergegeven uitgaat, kunt u dat programmeren. Een ander soort hypotheek betekent eveneens een geringe wijziging. De rente en de aflossing per jaar moeten dan anders worden berekend. Het voorbeeldprogramma is uiteraard niet bedoeld om daarop uw keuze te baseren. Voor een lineaire hypotheek geeft zij wel een redelijke schatting. Maar factoren op langere termijn als inflatie, belastingen en salaris zijn niet eenvoudig te voorzien.
Het is wel belangrijk op te merken dat een computeroverzicht door de te kiezen aannames sterk wordt beïnvloed. Een overzicht met verkeerde of gewenste aannames is in feite zinloos. U bent zo langzamerhand toch wel zo ver gevorderd, dat u begrijpt dat de kreet 'de computer zegt het' voor u niet als overtuigend bewijs mag gelden. Het programma drukken we nu af in afbeelding 11.15. Er is voldoende commentaar in verwerkt, zodat u met de bovenstaande opmerkingen erbij dit programma zou moeten kunnen begrijpen.

Ook hier kan het stroomdiagram erg eenvoudig zijn. We raden u aan bij wijze van oefening het stroomdiagram van het programma zelf te construeren. We laten het resultaat van een berekening eerst zien als we geen inflatie veronderstellen in afbeelding 11.16.

We zien dat het netto te betalen bedrag hoger is dan het bruto te betalen bedrag als de huurwaarde van het huis (hier *f* 117Ø,—) groter wordt dan de aftrekbare rente. Dit treedt op in het 29e jaar.
Als we 7% inflatie veronderstellen wordt het overzicht weergegeven in afbeelding 11.17.

Alle geldwaarden zijn herleid naar de guldenwaarde van dit moment. Door de inflatie worden de te betalen geldbedragen lager. De huurwaarde (niet inflatoir) overtreft nu de rente reeds in het 22e jaar.

```
LIST
10   REM LINEAIRE HYPOTHEEK
20   REM EERST DE DIMENSIES VAN DE HUURWAARDE-
30   REM EN BELASTINGSCHIJVEN OPGEVEN.
40   DIM HUURWAARDET(10,2),SCHIJF(10,3)
50   REM NU DE NOODZAKELIJKE GEGEVENS OPVRAGEN
60   INPUT "HYPOTHEEKBEDRAG";HYP
70   INPUT "RENTEPERCENTAGE IN PROCENTEN PER JAAR";RP
80   INPUT "LOOPTIJD HYPOTHEEK IN JAREN";LOOPT
90   INPUT "AANTAL TERMIJNEN PER JAAR";AT
100  INPUT "KOOPWAARDE VAN HET HUIS";KOOPW
110  INPUT "JAARPREMIE EXTRA LEVENSVERZEKERING IVM HUIS";LVP
120  INPUT "VASTE KOSTEN VOOR HET HUIS PER JAAR";VASTEK
130  INPUT "INFLATIEPERCENTAGE PER JAAR";INFLP
140  PRINT "BELASTBARE SOM INKOMSTENBELASTING,WAARBIJ"
150  PRINT "ALLE NIET HUISGEBONDEN AFTREKPOSTEN REEDS ZIJN"
160  INPUT "AFGETROKKEN.";BELSOM
170  REM BEREKEN DE WERKELIJKE JAARRENTE
180  RPJ=RP/(AT*100)
190  RPJ=-1+(1+RPJ)↑AT
200  PRINT "WERKELIJKE JAARRENTE IS";RPJ*100;" PROCENT"
210  REM BEREKEN HET INFLATIEPERCENTAGE
220  LET INFL=1/(1+INFLP/100)
230  REM NU DE HUURWAARDETABEL EN DE SCHIJVEN INLEZEN"
240  FOR R=0 TO 1
250  FOR K=0 TO 9
260  READ HUURWAARDET(K,R)
270  NEXT K
280  NEXT R
290  FOR R=0 TO 2
300  FOR K=0 TO 9
310  READ SCHIJF(K,R)
320  NEXT K
330  NEXT R
340  REM BEREKEN NU DE HUURWAARDE
350  REM DE ECONOMISCHE WAARDE IS GESCHAT ALS 0.7*KOOPWAARDE
360  REM EERSTE KOLOM VAN HUURWAARDET GEEFT DE ECONOMISCHE
370  REM WAARDE IN DUIZENDEN GULDENS.
380  REM DE TWEEDE KOLOM DE BIJBEHORENDE HUURWAARDE.
390  FOR K=9 TO 0 STEP -1
400  IF (KOOPW*7E-4)>=HUURWAARDET(K,0) THEN HUURWAARDE=HUURWAARDET(K,1)
410  NEXT K
420  REM NU HET OVERZICHT PRODUCEREN
430  PRINT
440  PRINT "OVERZICHT LINEAIRE HYPOTHEEK"
450  PRINT "JAAR","BRUTO","NETTO","SCHULD","RENTE"
460  LET SCHULD=HYP
470  LET AFLOS=HYP/LOOPT
480  FOR K=1 TO LOOPT
490  REM BRUTO KOSTEN ZIJN DE RENTE+AFLOSSING
500  REM +EXTRA LEVENSVERZEKERING+VASTE KOSTEN.
510  REM DE TE BETALEN RENTE IS OVER DE RESTERENDE SCHULD
520  LET RENTE=RPJ*SCHULD
```

Afb. 11.15.

```
530   LET BRUTO=RENTE+AFLOS+LVP+VASTEK
540   REM BEREKEN DE BELASTING ZONDER HUIS
550   LET INK=BELSOM
560   GOSUB 790
570   BELASTZH=B
580   REM NU DE BELASTING MET HUIS
590   LET INK=BELSOM+HUURWAARDE-RENTE
600   GOSUB 790
610   REM NETTO BEDRAG IS BRUTO MINUS BELASTNGAFTREK
620   LET NETTO=BRUTO-(BELASTZH-B)
630   LET SCHULD=SCHULD-AFLOS
640   REM NU AFRONDEN OP 2 DECIMALEN
650   LET BRUTOF=INT(0.5+100*BRUTO)/100
660   LET NETTOF=INT(0.5+100*NETTO)/100
670   LET SCHULDF=INT(0.5+100*SCHULD)/100
680   LET RENTEF=INT(0.5+100*RENTE)/100
690   PRINT K,BRUTOF,NETTOF,SCHULDF,RENTEF
700   REM DE LEVENSVERZEKERINGSPREMIE EN DE SCHULD STELLEN
710   REM WE AAN INFLATIE ONDERHEVIG.
720   LET LVP=LVP*INFL
730   LET SCHULD=SCHULD*INFL
740   LET AFLOS=AFLOS*INFL
750   REM INKOMSTEN,HUURWAARDE EN BELASTINGSCHIJVEN STELLEN
760   REM WE NIET INFLATOIR.
770   NEXT K
780   GOTO 1010
790   REM SUBROUTINE VOOR DE BELASTING BEREKENING
800   REM INPUT:INK ALS BELASTBARE SOM
810   REM :SCHIJF ALS SCHIJVENTARIEF
820   REM DE EERSTE KOLOM VAN DE SCHIJF GEEFT DE BELASTBARE SOM
830   REM DE TWEEDE HET TE BETALEN BEDRAG TOT AAN DIE SCHIJF
840   REM DE DERDE HET BELASTINGPERCENTAGE IN DE SCHIJF
850   REM OUTPUT:B ALS BETAALDE BELASTING
860   LET L=0
870   IF INK>SCHIJF(L,0) GOTO 900
880   LET L=L+1
890   GOTO 870
900   LET B=(INK-SCHIJF(L,0))*SCHIJF(L,2)+SCHIJF(L,1)
910   RETURN
920   REM NU DE DATA-STATEMENTS
930   REM EERST DE 2 KOLOMMEN VAN DE HUURWAARDETABEL
940   REM DAN DE 3 KOLOMEN VAN DE BELASTINGSCHIJVEN
950   REM DAN DE 3 KOLOMEN VAN DE BELASTINGSCHIJVEN
960   DATA 360,300,240,180,120,90,60,30,15,0
970   DATA 4680,3900,3120,2340,1560,1170,780,390,150,0
980   DATA 175126,120178,92703,70379,50410,34854,25014,14409,7568,0
990   DATA 101337,62874,44466,30179,18398,10620,6684,3291,1513,0
1000  DATA 0.72,0.7,0.67,0.64,0.59,0.5,0.4,0.32,0.26,0.2
1010  END
*
```

Vervolg afb. 11.15.

```
RUN
HYPOTHEEKBEDRAG
?150000
RENTEPERCENTAGE IN PROCENTEN PER JAAR
?10
LOOPTIJD HYPOTHEEK IN JAREN
?30
AANTAL TERMIJNEN PER JAAR
?12
KOOPWAARDE VAN HET HUIS
?150000
JAARPREMIE EXTRA LEVENSVERZEKERING IVM HUIS
?750
VASTE KOSTEN VOOR HET HUIS PER JAAR
?1500
INFLATIEPERCENTAGE PER JAAR
?0
BELASTBARE SOM INKOMSTENBELASTING,WAARBIJ
ALLE NIET HUISGEBONDEN AFTREKPOSTEN REEDS ZIJN
AFGETROKKEN.
?30000
WERKELIJKE JAARRENTE IS 10.4715 PROCENT
OVERZICHT LINEAIRE HYPOTHEEK
```

JAAR	BRUTO	NETTO	SCHULD	RENTE
1	22957.2	17907	145000	15707.2
2	22433.6	17551	140000	15183.6
3	21910	17194.9	135000	14660
4	21386.5	16838.9	130000	14136.5
5	20862.9	16482.9	125000	13612.9
6	20339.3	16126.8	120000	13089.3
7	19815.7	15770.8	115000	12565.7
8	19292.2	15414.8	110000	12042.2
9	18768.6	15058.8	105000	11518.6
10	18245	14702.7	100000	10995
11	17721.4	14346.7	95000	10471.5
12	17197.9	13990.7	90000	9947.88
13	16674.3	13634.7	85000	9424.31
14	16150.7	13278.6	80000	8900.73
15	15627.2	12922.6	75000	8377.16
16	15103.6	12566.6	70000	7853.59
17	14580	12210.5	65000	7330.02
18	14056.4	11854.5	60000	6806.44
19	13532.9	11498.5	55000	6282.87
20	13009.3	11173.6	50000	5759.3
21	12485.7	10859.4	45000	5235.73
22	11962.2	10545.3	40000	4712.15
23	11438.6	10231.2	35000	4188.58
24	10915	9917.01	30000	3665.01
25	10391.4	9602.86	25000	3141.44
26	9867.86	9288.72	20000	2617.86
27	9344.29	8974.57	15000	2094.29
28	8820.72	8660.43	10000	1570.72
29	8297.15	8346.29	5000	1047.15
30	7773.57	8032.14	0	523.57

```
END PROGRAM
*
```

Afb. 11.16.

```
 RUN
HYPOTHEEKBEDRAG
?150000
RENTEPERCENTAGE IN PROCENTEN PER JAAR
?10
LOOPTIJD HYPOTHEEK IN JAREN
?30
AANTAL TERMIJNEN PER JAAR
?12
KOOPWAARDE VAN HET HUIS
?150000
JAARPREMIE EXTRA LEVENSVERZEKERING IVM HUIS
?750
VASTE KOSTEN VOOR HET HUIS PER JAAR
?1500
INFLATIEPERCENTAGE PER JAAR
?7
BELASTBARE SOM INKOMSTENBELASTING,WAARBIJ
ALLE NIET HUISGEBONDEN AFTREKPOSTEN REEDS ZIJN
AFGETROKKEN.
?30000
WERKELIJKE JAARRENTE IS 10.4715 PROCENT
OVERZICHT LINEAIRE HYPOTHEEK
JAAR          BRUTO        NETTO        SCHULD       RENTE
 1            22957.2      17907        145000       15707.2
 2            21064.1      16499.3      130841       14190.3
 3            19326.9      15205.6      117914       12804.6
 4            17733.3      14016.7      106119       11539.6
 5            16271.8      12924.7      95361.9      10385.2
 6            14932.2      11921.9      85558.3      9332.5
 7            13704.5      11001.3      76629.3      8373.08
 8            12580.1      10156.4      68502.4      7499.25
 9            11550.5      9381.34      61110.9      6703.92
 10           10608.2      8683.95      54393.3      5980.56
 11           9746.16      8084.9       48293.1      5323.15
 12           8957.94      7535.40      42758.3      4726.16
 13           8237.57      7031.77      37741        4184.5
 14           7579.53      6570.13      33197.1      3693.48
 15           6978.75      6147.23      29086.3      3248.8
 16           6430.56      5759.96      25371.2      2846.5
 17           5930.65      5405.48      22017.7      2482.93
 18           5475.04      5081.15      18994.4      2154.74
 19           5060.09      4784.54      16272.5      1858.87
 20           4682.41      4513.42      13825.4      1592.49
 21           4338.92      4265.71      11628.8      1353.01
 22           4026.74      4039.52      9660.51      1138.04
 23           3743.26      3833.1       7899.95      945.42
 24           3486.06      3644.81      6328.39      773.12
 25           3252.91      3473.18      4928.66      619.32
 26           3041.77      3316.83      3684.98      482.34
 27           2850.75      3174.5       2582.93      360.63
 28           2678.12      3045.01      1609.3       252.78
 29           2522.3       2927.31      752.01       157.49
 30           2381.83      2820.39      0            73.59
END PROGRAM
*
```

Afb. 11.17.

11.6. Annuïteitshypotheek

Voor een annuïteitshypotheek geldt dat per termijn een vast bedrag voor rente plus aflossing wordt betaald gedurende de hele looptijd van de hypotheek.
Stellen we het rente-percentage R en het aantal termijnen in de looptijd M, dan betalen we per termijn het hypotheekbedrag gedeeld door een annuïteitsfactor, die luidt:

$$\frac{1}{1+R} \times \frac{1-\dfrac{1}{(1+R)^M}}{1-\dfrac{1}{1+R}}$$

We drukken nu een programma (afb. 11.18) af met zijn resultaat, dat weergeeft hoeveel gulden per maand moet worden betaald per ƒ 1ØØØ,— opgenomen hypotheek. Het geproduceerde overzicht geeft looptijden van 2Ø, 25 en 3Ø jaar weer voor werkelijke rentepercentages van 7 tot 12% (met stappen van Ø.5%).

```
*LIST
10    REM ANNUITEITSHYPOTHEEK
20    PRINT "LOOPTIJD","RENTEPERC.","PER FL 1000 PER MAAND"
30    FOR LOOPT=20 TO 30 STEP 5
40    FOR RP=7 TO 12 STEP 0.5
50    LET R=RP/(12*100)
60    LET F1=1/(1+R)
70    LET F2=1-1/((1+R)↑(LOOPT*12))
80    LET FACTOR=F1*F2/(1-F1)
90    PRINT LOOPT,RP,1000/FACTOR
100   NEXT RP
110   NEXT LOOPT
120   END
*RUN
LOOPTIJD    RENTEPERC.   PER FL 1000 PER MAAND
 20          7            7.753
 20          7.5          8.05591
 20          8            8.36437
 20          8.5          8.67814
 20          9            8.99723
 20          9.5          9.32135
 20          10           9.6502
 20          10.5         9.98379
 20          11           10.3219
 20          11.5         10.6643
 20          12           11.0109
 25          7            7.0678
 25          7.5          7.3899
 25          8            7.71814
 25          8.5          8.05219
 25          9            8.39195
```

Afb. 11.18.

```
25          9.5         8.737
25          1Ø          9.Ø87
25          1Ø.5        9.44181
25          11          9.8Ø117
25          11.5        1Ø.1647
25          12          1Ø.5322
3Ø          7           6.653Ø4
3Ø          7.5         6.99215
3Ø          8           7.33762
3Ø          8.5         7.689Ø7
3Ø          9           8.Ø4623
3Ø          9.5         8.4Ø858
3Ø          1Ø          8.77572
3Ø          1Ø.5        9.14738
3Ø          11          9.52328
3Ø          11.5        9.9Ø293
3Ø          12          1Ø.2861
END PROGRAM
*
```

Samenvatting hoofdstuk 11

In dit hoofdstuk hebben we verschillende BASIC programma's uitgewerkt. Zij dienen als voorbeeld hoe in BASIC zou kunnen worden geprogrammeerd.

Multiple choice-vragen bij hoofdstuk 11

A. a. Als we in het stuitende bal programma regel 130 vervangen door
130 AFSTAND NU =(OORSPRONG + STUIT) * (VERH ↑ K)
blijft het programma dezelfde resultaten produceren.

b. Als we in het stuitende bal programma het statement op regel 130 vervangen door
(OORSPRONG + STUIT) * (1 + VERH) ↑ K – 1
blijft het programma dezelfde resultaten produceren.

1 uitspraak a is goed en uitspraak b is goed
2 uitspraak a is fout en uitspraak b is goed
3 uitspraak a is goed en uitspraak b is fout
4 uitspraak a is fout en uitspraak b is fout

B. a. Het stuitende bal programma geeft hetzelfde resultaat als de volgende wijzigingen worden aangebracht:
```
 95 LET K = 0
100 LET K = K + 1
105 IF K > KEER GOTO 150
140 GOTO 100
```
b. Als in het stuitende bal programma (grafische weergave) de regels 110 en met 130 worden weggelaten, verdwijnt de horizontale rij punten uit het geprinte resultaat.

1 uitspraak a is goed en uitspraak b is goed
2 uitspraak a is fout en uitspraak b is goed
3 uitspraak a is goed en uitspraak b is fout
4 uitspraak a is fout en uitspraak b is fout

C. a. Als in TAB(X) de variabele X een gebroken getal is, rondt BASIC X voor gebruik af tot een geheel getal.

b. In het programma dat het stuitende bal printen grafisch weergeeft had regel 150 ook mogen zijn FOR K = 0 TO KEER – 1 zonder dat het resultaat verandert.

1 uitspraak a is goed en uitspraak b is goed
2 uitspraak a is fout en uitspraak b is goed
3 uitspraak a is goed en uitspraak b is fout
4 uitspraak a is fout en uitspraak b is fout

D. a. Wanneer de gebruiker in het getal raden programma zowel voor het maximale getal als voor het raden alleen gebroken getallen intypt met een fractie ≠ Ø, kan hij nog steeds juist raden.

b. Wanneer in het getal raden programma wordt toegevoegd 75 RAAD = ABS (INT(RAAD)) verandert dit het resultaat van het raden niet, vergeleken met het raden met positieve gehele getallen in het oorspronkelijke programma.

1 uitspraak a is goed en uitspraak b is goed
2 uitspraak a is fout en uitspraak b is goed
3 uitspraak a is goed en uitspraak b is fout
4 uitspraak a is fout en uitspraak b is fout

E. a. Bij het vergelijken van strings wordt de kortste string achter aangevuld gedacht met blanko's.
b. Bij het vergelijken van getallen worden deze afgerond tot gehele getallen.

1 uitspraak a is goed en uitspraak b is goed
2 uitspraak a is fout en uitspraak b is goed
3 uitspraak a is goed en uitspraak b is fout
4 uitspraak a is fout en uitspraak b is fout

F. a. In het bioritme programma zou regel 110 zonder verandering van het resultaat veranderd kunnen worden in FOR K = 0 TO 11.
b. In het bioritme programma zouden zonder verandering van het resultaat de regels 80 en 90 vervangen kunnen worden door: 80 INPUT 'TYPE MAAND EN JAAR VAN HET OVERZICHT'; 0MAAND, 0JAAR

1 uitspraak a is goed en uitspraak b is goed
2 uitspraak a is fout en uitspraak b is goed
3 uitspraak a is goed en uitspraak b is fout
4 uitspraak a is fout en uitspraak b is fout

G. a. Wanneer in het bioritme programma de regels 160, 170 en 180 worden vervangen door:
160 D = GDAG : M = GMAAND : J = GJAAR
is het resultaat van het programma hetzelfde.
b. Wanneer in het bioritme programma de berekeningssubroutine niet was gebruikt, maar de berekening, waar nodig, direct was uitgevoerd, zou een programma met minder regels zijn ontstaan.

1 uitspraak a is goed en uitspraak b is goed
2 uitspraak a is fout en uitspraak b is goed
3 uitspraak a is goed en uitspraak b is fout
4 uitspraak a is fout en uitspraak b is fout

H. Bij vergroting van het aantal schijven in de belastingschijven dienen in het lineaire hypotheek programma zeker de volgende regels gewijzigd te worden:

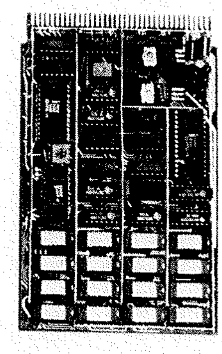
1 geen wijzigingen nodig
2 regels 40, 250, 960 en 970
3 regels 40, 300, 860, 900, 980, 990 en 1000
4 regels 40, 300, 980, 990 en 1000

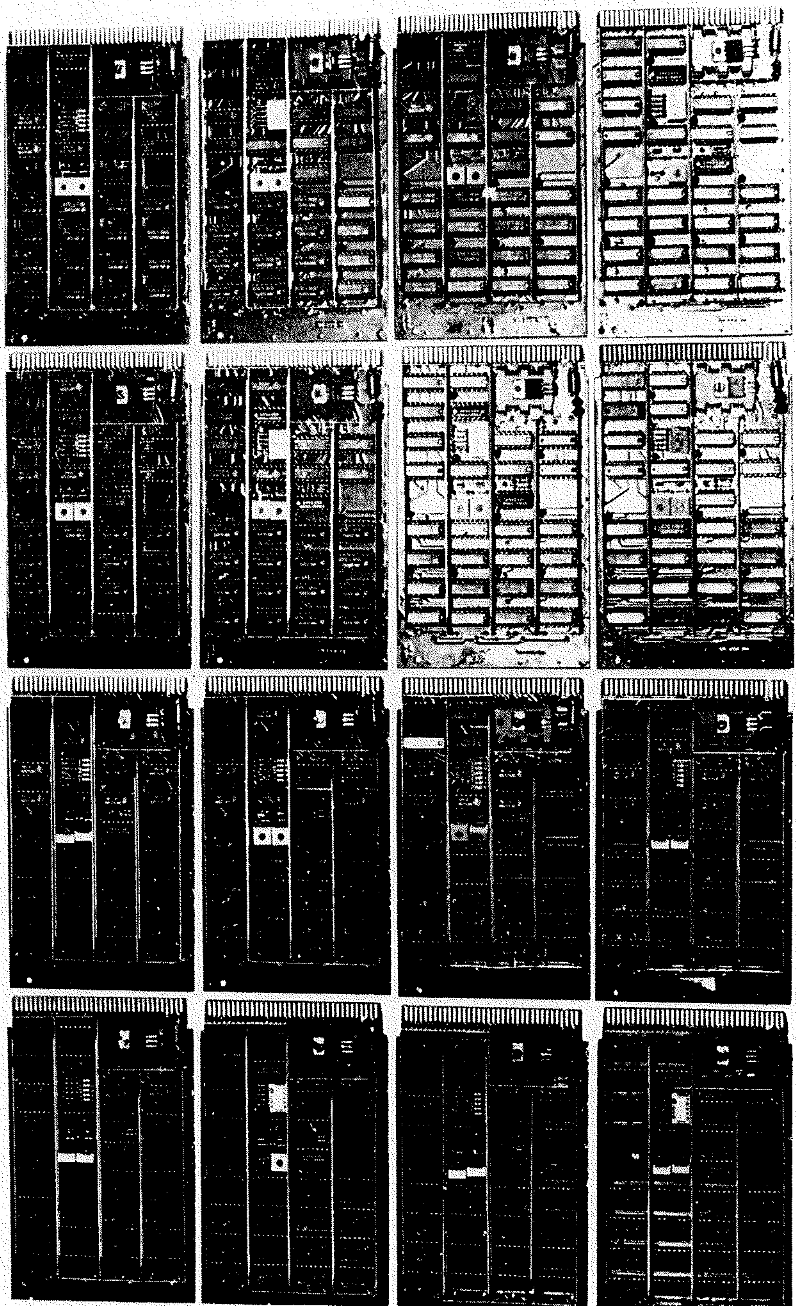
I. a. In het lineaire hypotheek programma zijn de te printen bedragen afgerond op 2 decimalen; dus centen (regel 650 t/m 680). Bij afronding op guldens zou regel 650 moeten zijn BRUTOF = INT (BRUTO) + 0.5.
b. Wanneer in het lineaire hypotheek programma een andere hypotheekvorm wordt geprogrammeerd veranderen in elk geval de regels 460, 470, 520, 530, en 630.

1 uitspraak a is goed en uitspraak b is goed
2 uitspraak a is fout en uitspraak b is goed
3 uitspraak a is goed en uitspraak b is fout
4 uitspraak a is fout en uitspraak b is fout

J. Stel dat in de belastingschijven de hoogste schijf verdwijnt in regelnr. 980, 990 en 1000 terwijl de overige schijfwaarden gelijk blijven. Op welke regels moet het lineaire hypotheek programma dan zeker gewijzigd worden om foutmeldingen te voorkomen.

1 geen wijziging nodig
2 regels 250, 960 en 970
3 regels 300, 860, 900, 980, 990 en 1000
4 regels 300, 980, 990 en 1000





De kaart (links) bevat 32 k bytes geheugen en vervangt de 16 kaarten rechts, die elk 2 k bytes bevatten. Deze foto laat zien wat drie jaar in de micro-elektronica doen. (Geheugenkaarten microcomputersysteem van de vakgroep Digitale Systemen, TH Eindhoven).

Hoofdstuk 12

BASIC en ASSEMBLER

In dit hoofdstuk zullen we aangeven hoe in principe een interpreter werkt. In het Nederlands wordt een interpreter wel vertolker genoemd. De bedoeling is om een beter begrip te wekken voor wat er in feite bij executie van een BASIC programma gebeurt. Daarnaast wordt de programmering in ASSEMBLER taal behandeld. Dit hoofdstuk geeft een overzicht van de instructies in ASSEMBLER taal, die de 8080 microprocessor in onze Teleac/DAI personal computer kan verwerken. Een en ander wordt met enige voorbeelden toegelicht.

12.1. BASIC interpreter

Veronderstel dat we een BASIC programma schrijven. Daarna willen we dit laten executeren. In feite moet dan de BASIC interpreter de BASIC statements naar de machine-instructies van de computer vertalen. De interpreter doet dit regel voor regel. De BASIC interpreter bestaat uit verschillende onderdelen. Hij is zelf via de ASSEMBLER taal in machinetaal geprogrammeerd en permanent in het ROM-geheugen (ready only memory = alleen leesbaar geheugen) van de computer aanwezig. Zoals in hoofdstuk 9 bij de programma-opbouw reeds is betoogd is een opbouw in duidelijk gedefinieerde delen nodig. In het algemeen kunnen we bij een BASIC interpreter de volgende delen onderscheiden:

- *Een hoofdprogramma,* dat bepaalt wat er steeds moet gebeuren. Zodra BASIC is geladen, komt dit hoofdprogramma in actie. Alle andere programmadelen zijn ondergeschikt aan het hoofdprogramma. Zij worden door dit programma aangeroepen. Na afloop van zo'n programmadeel wordt weer naar het hoofdprogramma teruggekeerd.
- *Een regel editor.* Dit programma maakt het mogelijk om een BASIC programma in te typen. Het editor programma voert in de eerste plaats het beheer van de mogelijkheden om een regel in te typen en kleine typefouten te verbeteren. Het programma heeft ook tot taak de ingevoerde statements op zo effectief mogelijke wijze op te slaan. Een regel wordt namelijk zo opgeslagen dat hij zo weinig mogelijk geheugenruimte inneemt en later gemakkelijk kan worden geëxecuteerd. Ieder statement zal dus niet met zijn volledige naam worden opgeslagen, maar als een gecodeerde waarde. Meestal controleert deze editor ook nog of er BASIC statements worden ingevoerd, waarin duidelijk tegen de regels wordt gezondigd.

In dit hoofdstuk zijn de regelnummers weer met nullen (ØØ) op 'computerwijze' afgedrukt.

– *Een programma editor.* Deze heeft tot taak de regels van de regel editor op een zodanige plaats in het geheugen te zetten dat ze later door de rest van de BASIC interpreter weer kunnen worden teruggevonden.

– *Een regelexecutie routine.* Deze zal bij executie van een regel uit een BASIC programma de regel ontleden. Daarbij wordt nagegaan of de BASIC statements op toegestane wijze zijn gebruikt, of de waarden in het toegestane bereik liggen, enz. De regelexecutie routine is uiteraard ook verantwoordelijk voor het produceren van foutmeldingen bij een eventuele foutieve regel. Een deel van die fouten kan al in de vorige regel gesignaleerd zijn. Voor ieder statement wordt de bijbehorende routine aangeroepen.

– *Een regelvolgorde routine.* De taak hiervan is steeds te bepalen welke regel moet worden geëxecuteerd bij de volgende executiestap. Normaal is dat steeds de volgende regel. Bij sprongen treden echter afwijkingen van deze regelmaat op en dan moet er eventueel met een andere regel van het gebruikersprogramma worden verder gegaan.

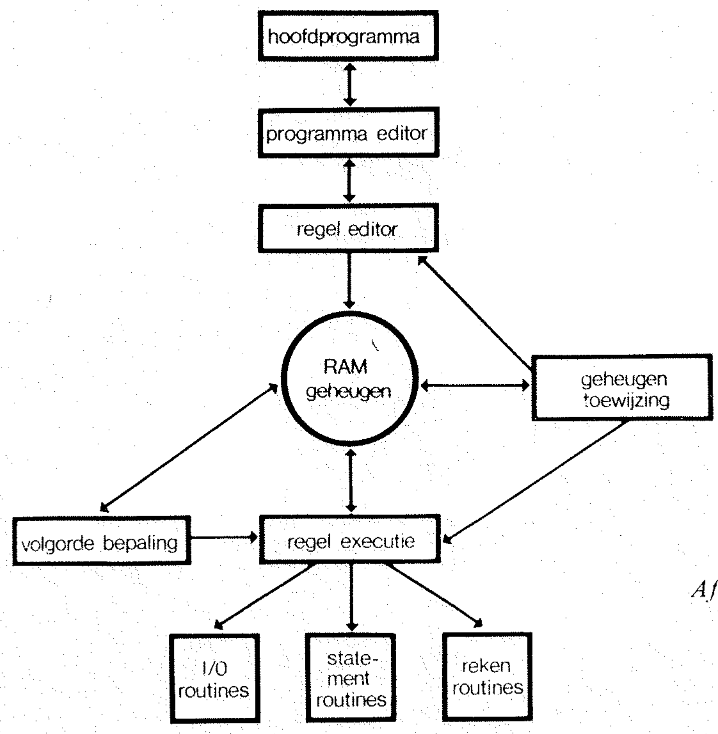
– *Rekenroutines.* In BASIC kunnen we een veelheid van berekeningen uitvoeren. U weet dat de processor van de Teleac/DAI personal computer slechts op 8 bits tegelijk opereert. Wanneer dus met grote getallen moet worden gerekend, zijn daarvoor meerdere instructies nodig. Er moet dan worden vastgelegd hoe groot de te berekenen getallen mogen zijn. Het kernpunt hier is uiteraard het formaat van die getallen. Dat wil zeggen, hoe staan de getallen in het geheugen. We hebben in onze computer twee soorten getallen: *integers* (gehele getallen) en *floating point* (gebroken getallen). Zij hebben een verschillend formaat. De rekenroutines voor de integers verschillen dan ook van die voor de floating point getallen.

– *I/O routines.* Voor ieder van de aangesloten randapparaten moet er een routine aanwezig zijn, die het randapparaat bestuurt. In hoofdstuk 19 zullen we onder andere enkele voorbeelden van zulke routines zien.

– *De statement routines.* Voor ieder statement, dat in BASIC mogelijk is, moet er in de computer een routine aanwezig zijn. In die routine wordt eerst gecontroleerd of de bij deze routine behorende invoergegevens in de juiste vorm aanwezig zijn. De executie van het statement, dat tot het door de programmeur gewenste resultaat leidt, wordt uiteraard ook hier verricht. Behalve statement routines zijn er uiteraard ook nog de routines voor commando's en operatoren.

– *Een geheugentoewijzing routine.* Deze laatste routine zorgt er voor dat het geheugen steeds zo goed mogelijk wordt benut. De routine wijst de plaats aan waar variabelen worden gezet, enz.

Er is in de BASIC interpreter een groot aantal routines aanwezig. De bovenstaande hoofddelen geven slechts een indruk van de opbouw. Het programmeren van de interpreter is dan ook een complexe zaak (zie afbeelding 12.1).



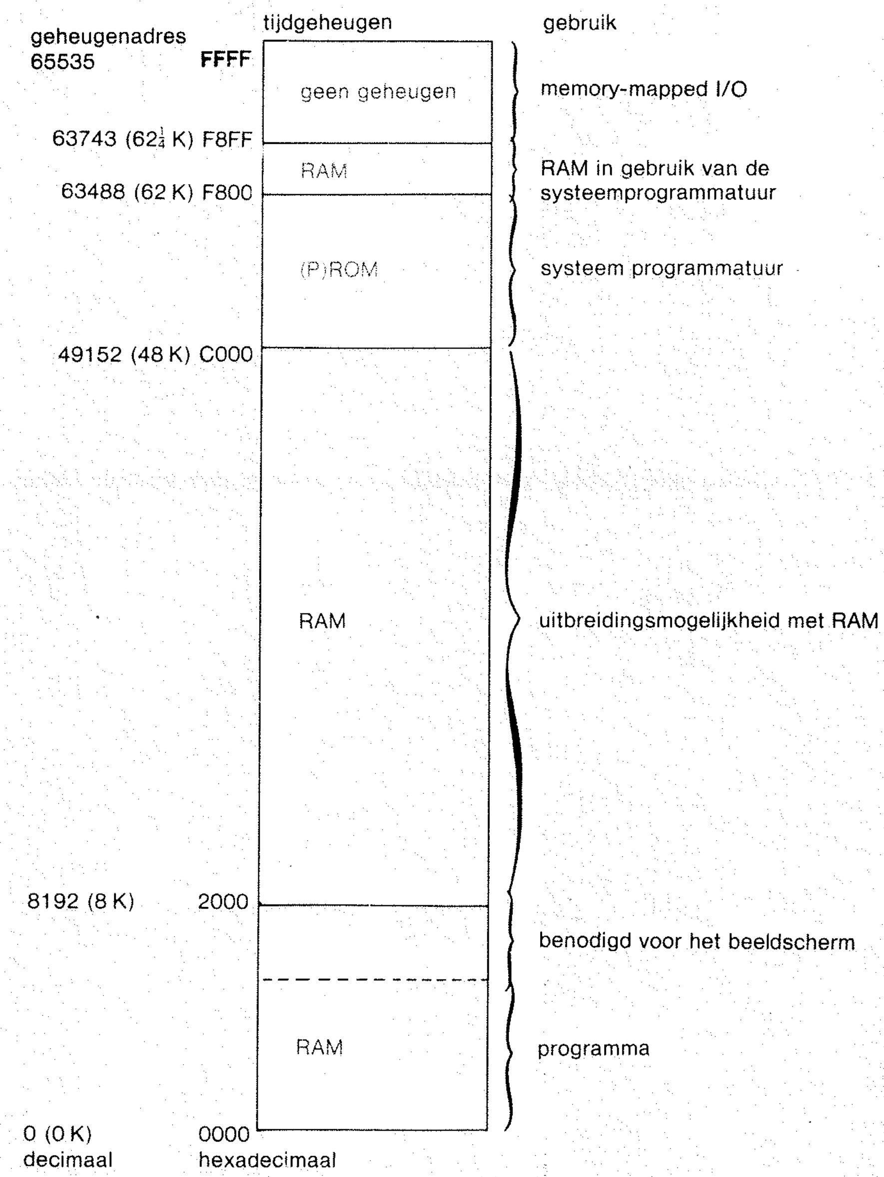
Afb. 12.1. Enkele routines uit een interpreter.

Eigenlijk hoeven alleen de variabelen, enz. in het RAM geheugen (waarin kan worden gelezen en geschreven) aanwezig te zijn. Ze kunnen namelijk bij de programma-executie van waarde veranderen. Het BASIC programma zelf bevindt zich ook in het RAM geheugen. Immers, in het geval van het Teleac/DAI-systeem, heeft de gebruiker het programma ingetypt of vanaf de cassetteband ingevoerd. In principe zou het programma zelf wel in (P)ROM mogen staan.
De BASIC interpreter kan in het (P)ROM geheugen (waarin alleen kan worden gelezen) van het systeem worden geplaatst. De routines veranderen namelijk niet meer. Bij de Teleac/DAI personal computer is dit dan ook gedaan. De interpreter beslaat ongeveer 17 K bytes (P)ROM geheugen (*NB:* 1K = 1Ø24 bytes). U ziet een omvangrijk programma. Daarnaast is er dan nog 5K bytes in (P)ROM aanwezig om de I/O routines en andere hulproutines te verzorgen.
N.B. Door een hardwareschakeling worden de PROM's beurtelings aangeschakeld zodat de adresruimte van het PROM geheugen 14 K bedraagt.

12.2. Geheugenindeling

Wanneer er in ASSEMBLER wordt geprogrammeerd, moet uiteraard bekend zijn welke geheugendelen RAM bevatten en waar

de (P)ROM aanwezig is. Verder mag de ASSEMBLER programmeur geen veranderingen aanbrengen in RAM-gedeeltes die door de BASIC interpreter gebruikt worden. Een gebruikelijke manier om dit aan te geven is de zogenaamde *memory-map*. Daarin wordt van het geheugen aangegeven welke soort het is (RAM, (P)ROM, memory-mapped I/O) en welk programma het mag gebruiken. Voor de Teleac/DAI personal computer is de memory-map weergegeven in afbeelding 12.2.
Daaruit blijkt, dat in ieder Teleac/DAI-systeem minimaal 8K RAM geheugen aanwezig is. De BASIC programma's en de variabelen worden daarin geplaatst. Vanaf het hoogste RAM adres hiervan wordt naar beneden toe het geheugen gebruikt voor het genereren van het beeld op het beeldscherm. De hoeveelheid RAM die daarvoor wordt gebruikt, is bepaald door het feit of het scherm met tekst



Afb. 12.2. Memory-map van de Teleac/DAI personal computer.

of met grafische informatie wordt beschreven. Nadere uitwerking hiervan wordt gegeven in hoofdstuk 18.

Wordt het RAM geheugen uitgebreid bijvoorbeeld tot 48K, dan schuift dit beeldschermgebied met de bovengrens mee.
In het (P)ROM gebied (48-62K) is alle systeemprogrammatuur van het systeem aanwezig (BASIC interpreter, utilities, I/O drivers, enz.). De systeemprogrammatuur gebruikt zelf nog een 1/4 K RAM gebiedje (vanaf 62K).
De adressen boven 63743 (F8FF Hex) zijn in gebruik voor de memory-mapped-I/O. Input en output poorten gedragen zich dus in dit gebied als geheugenplaatsen. Deze I/O wordt nader uitgewerkt in hoofdstuk 17.

We kunnen een aantal hulpmiddelen onderscheiden, die de BASIC interpreter nodig heeft om zijn werk te kunnen verrichten. In de eerste plaats is dat natuurlijk een tabel, waarin alle gereserveerde woorden voorkomen, die zijn toegestaan voor BASIC zoals bij voorbeeld LIST, ABS, enz. Daarnaast is een tabel nodig, die vastlegt welke variabele namen zijn gebruikt en waar die variabelen zich in het geheugen bevinden. Verder is er in het RAM geheugen ook een *regelbuffer* nodig. Deze regelbuffer is een stuk geheugen, waar de regel, die wordt ingetypt, geplaatst wordt. Tenslotte is er voor het reken- en ontleedwerk van de interpreter RAM geheugen nodig. We zien hier dus dat de interpreter behalve een ROM geheugen, ook ruimte nodig heeft in het RAM geheugen om goed te kunnen werken. Verder heeft de interpreter ruimte in RAM nodig voor de stack om de routine aanroepen correct af te handelen.

12.3. BASIC-ASSEMBLER koppeling

Er moeten dus afspraken worden gemaakt welk deel in RAM door de interpreter gebruikt wordt en welk deel door de programmeur mag worden benut. Wanneer programma's geheel in BASIC worden geschreven, is dit geen probleem. De BASIC interpreter, die voor de hele afhandeling zorg draagt, kent de afspraken en houdt zich daar ook aan. Het begrip 'geheugenplaats' komt normaal in BASIC als zodanig niet voor. De BASIC programmeur kan de interpreter opgeven hoeveel ruimte in RAM gebruikt mag worden voor array's en strings te zamen. Daartoe dient het statement CLEAR.
CLEAR 1ØØ betekent dat 1ØØ bytes mogen worden gebruikt om array's en strings in op te bergen. Het getal achter CLEAR moet liggen tussen 4 en 32767.
CLEAR zet alle array elementen op Ø en maakt alle string tot lege strings en kent daarna de gewenste ruimte toe.
Indien de foutmelding 'OUT OF STRING SPACE' of 'OUT OF MEMORY' optreedt kan worden getracht door middel van CLEAR meer ruimte te creëren. Deze ruimte gaat evenwel ten koste van de ruimte voor het programma zelf met de bijbehorende overige variabelen.
Gewoonlijk is het zo dat ook het eerste deel van het RAM geheugen bijvoorbeeld vanaf adres Ø tot adres 256 gereserveerd is voor het systeem zelf. Daar wordt dus het programma dan ook niet neergezet. Hier bestaan verschillen tussen de computers, zodat voor de details

```
*LIST
10    REM VERTALING NAAR HEXADECIMAAL
20    PRINT "DECIMAAL","HEXADECIMAAL"
30    FOR G%=0 TO #F
40    PRINT G%,HEX$(G%)
50    NEXT G%
60    END
*RUN
DECIMAAL    HEXADECIMAAL
 0          0
 1          1
 2          2
 3          3
 4          4
 5          5
 6          6
 7          7
 8          8
 9          9
 10         A
 11         B
 12         C
 13         D
 14         E
 15         F
END PROGRAM
*
*LIST
10    REM VERTALING NAAR HEXADECIMAAL
20    PRINT "DECIMAAL","HEXADECIMAAL"
30    FOR G%=#FF20 TO #FF2F
40    PRINT G%,HEX$(G%)
50    NEXT G%
60    END
*RUN
DECIMAAL    HEXADECIMAAL
 65312      FF20
 65313      FF21
 65314      FF22
 65315      FF23
 65316      FF24
 65317      FF25
 65318      FF26
 65319      FF27
 65320      FF28
 65321      FF29
 65322      FF2A
 65323      FF2B
 65324      FF2C
 65325      FF2D
 65326      FF2E
 65327      FF2F
END PROGRAM
*
```

Afb. 12.3.

weer de handleiding moet worden geraadpleegd. Maar we zullen in de volgende hoofdstukken zien dat er in microcomputersystemen BASIC statements zijn, die wel degelijk met het begrip geheugenplaats werken. Deze statements zijn toegevoegd om eenvoudiger in zo'n systeem te kunnen werken. Op dat moment moeten de afspraken ook aan de programmeur bekend zijn. Dit is nog nadrukkelijker het geval als we zoals hier, toestaan zelf geschreven programma's in machinetaal te koppelen aan BASIC programma's.
Nu is het in de praktijk zo (zie ook *Microprocessors 1*) dat de ASSEMBLER programmeur over listings van zijn programma beschikt, waarvan de adressen in hexadecimale waarde zijn weergegeven. Vandaar dat de Teleac/DAI personal computer in BASIC ook beschikt over een vertaling van hexadecimale naar decimale notatie. We laten daartoe een hexadecimaal getal voorafgaan door een #. Ook de omgekeerde vertaling van decimaal naar hexadecimaal is ingebouwd. Daartoe is het statement HEX$(X) aanwezig. X is een decimaal getal. HEX$(X) geeft de hexadecimale weergave van dit getal string. Zie afbeelding 12.3.

Voor het uittesten van programma's die geschreven zijn in machinetaal beschikken we over een zogenaamde monitor (ook utilities genoemd). Deze monitor stelt ons in staat na te gaan wat het machinetaal-programma precies doet. Veel van de commando's die we in de monitor kunnen aanroepen vragen om hexadecimale invoer of geven hexadecimale waarden als uitvoer af. We kunnen de monitor aanroepen met het BASIC commando UT. Dit staat voor UTilities. De computer antwoordt met de melding PC UTILITY V 3.3. (Dit staat voor Personal Computer Utilities versie 3.3.) Op de volgende regel wordt dan > neergezet ten teken dat op een commando gewacht wordt. In hoofdstuk 16 gaan we uitgebreid op de monitor in. In ons voorbeeld laten we de inhoud van de geheugenplaatsen CØØØ tot CØ2F (hexadecimaal) uitprinten. De regel begint met het hexadecimale adres van de adresplaatsen, die vervolgens op deze regel geplaatst worden (4 hexadecimale characters). Daarna volgen in groepjes van 2 hexadecimale characters de inhouden van de adresplaatjes. Bij adressen die een veelvoud van 16 zijn wordt automatisch op een nieuwe regel begonnen. Na het uitprinten komt weer de >. We typen in dit voorbeeld een B in om weer in BASIC te komen. BASIC geeft dan weer zijn melding BASIC V1.Ø.
In hoofdstuk 16 zullen we uitgebreider ingaan op de utility mogelijkheden. Nu hebben we uitsluitend beoogd een voorbeeld te geven van het gebruik van hexadecimale characters in afbeelding 12.4.

```
*UT
PC UTILITY V3.3
>DCØØØ CØ2F

CØØØ C3 19 C7 C3 35 CØ C3 F3 CØ C3 FB C1 C3 79 CØ C3
CØ1Ø BB CØ C3 D5 CØ C3 AC CØ C3 1E C2 C3 34 C2 C3 49
CØ2Ø C2 C3 61 C3 C3 73 C5 C3 B2 C5 C3 14 C6 C3 53 C6
>BBASIC V1.Ø
*
```

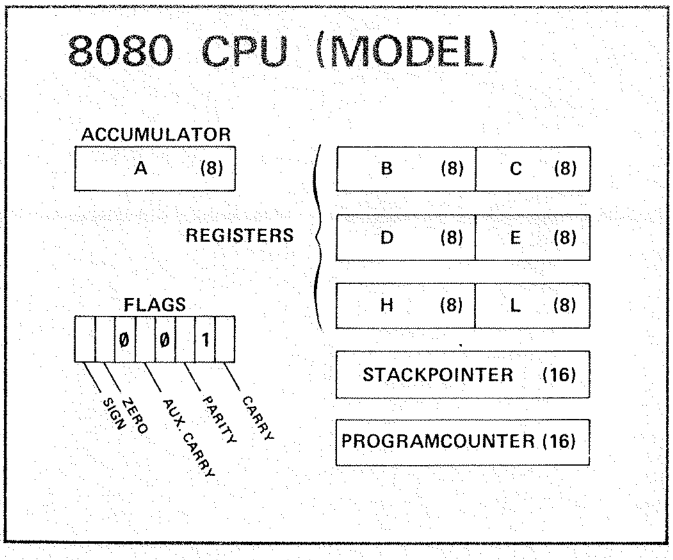
Afb. 12.4.

12.4. Uitgewerkt interpreter-voorbeeld

Aan de hand van een eenvoudig voorbeeld gaan we nu eens na wat er zoal gebeurt als we een regel executeren. We nemen daarvoor regel A = ABS(– 1). Voor het gemak gaan we er van uit dat de BASIC interpreter de voorbereidende handelingen om deze regel te vinden reeds heeft verricht. Onze te executeren regel wordt nu naar een regelbuffer gehaald, die de standaardplaats is vanwaar de regel wordt behandeld. In principe vinden nu de volgende handelingen plaats: De interpreter tracht uit te vinden of hij hier te maken heeft met een commando, een statement of een operator Hij doorloopt daarvoor de tabel met de toegestane BASIC woorden. Wanneer hij het gewenste woord vindt wordt de daarbij behorende actie ondernomen. De tabel bevat verder verwijzingen naar de routine, die de actie uitvoert. De regel wordt van links naar rechts gelezen. De begin-adressen van de routines, die bij de executie horen, worden op een stapel *(stack)* geplaatst. Het laatste adres wordt straks als eerste opgehaald. De executie werkt zo van rechts naar links, zoals we ook hadden kunnen verwachten. De argumenten voor de verschillende operaties worden op een andere stapel gezet *(argumentenstapel)*. Een routine haalt de invoerwaarden, die hij nodig heeft, van de top van de argumentenstapel. Het resultaat van de routine wordt op de top van de stack geplaatst voor gebruik door de volgende routine. Zo worden dus de tussenresultaten overgedragen.

In bovenstaand voorbeeld (A = ABS(– 1)) wordt dus eerst deze regel naar de regelbuffer gehaald. Nu zoekt hij naar het eerst voorkomende woord, dat voor BASIC is gereserveerd. In dit geval is de conclusie dat A niet in zijn tabel voorkomt en A blijft dus gewoon staan. Vervolgens wordt bekeken of = voorkomt. Dit is wel het geval en de = in de invoerbuffer wordt vervangen door een code voor de latere uitvoering van deze = bewerking. Nadat de regel op deze wijze is behandeld, wordt opnieuw het eerste karakter bezien (de letter A). Dit karakter hoorde niet bij de voor BASIC gereserveerde woorden in de tabel. De conclusie luidt nu: het LET statement is weggelaten, hetgeen overigens is toegestaan zoals we reeds in veel voorbeelden zagen. De interpreter roept nu de bij het LET statement horende routine aan. Daarin wordt de A en het daarop volgende karakter onderzocht. Wanneer dat laatste karakter een '(' is, hebben we hier te maken met een array. Er staat echter geen '(', maar een ' = ' zodat de conclusie is dat A een gewone variabele is. Deze (dus A) wordt in de lijst van de variabelen opgezocht. Wanneer hij voorkomt, wordt zijn bijbehorend adres opgehaald en in de stack geplaatst. Komt hij niet voor, dan wordt A toegevoegd aan de lijst van de variabele namen en het eerstvolgende vrije adres in de variabele ruimte wordt voor A gereserveerd. Nu wordt bekeken of achter de ' = ' weer een variabele staat. Dat is hier niet het geval. Staat er dan soms een constante? Ook niet! De interpreter gaat nu de mogelijkheid na dat er een ingebouwde functie wordt aangeroepen. Dit blijkt het geval te zijn, namelijk ABS. De routine die bij ABS hoort, wordt nu via de tabel van gereserveerde namen opgeroepen. Deze routine ontdekt het eerste haakje ')'. Om het verhaal niet al te lang en te gedetailleerd te maken: via de argumentenstapel wordt het resultaat 1 aan de variabele A toegekend.

In dit voorbeeld is slechts een globale indruk gegeven van wat er zoal in BASIC bij het verwerken van een eenvoudige regel gebeurt. Het is nu dan ook wel begrijpelijk dat er relatief veel tijd nodig is voor de executie van een BASIC programma via de interpreter. U hebt enig inzicht gekregen in de veelheid van handelingen, die de interpreter moet uitvoeren. Omdat u als gebruiker niet precies weet welke instructies allemaal afgehandeld zijn als effect van de BASIC instructie, kunt u niet precies weten hoe lang de executie van een BASIC statement duurt. Dat is namelijk afhankelijk van zaken als: welke argumenten zijn er in het statement gebruikt, enz. U merkt wel dat de computer bij gebruik van een hogere orde taal veel voor u doet.
Wanneer we machinecode instructies gebruiken, is wel precies vastgelegd hoeveel tijd iedere instructie in beslag neemt. In *Microprocessors 1* wordt in hoofdstuk 11 het overzicht gegeven welke machinecode-instructies er in de 8080 microprocessor voorkomen (zie de instructieset). Daarbij is een classificatie gemaakt naar de gegevens transportinstructies, de rekenkundige en de logische instructies, de programmabesturingsinstructies en de speciale instructies. We gaan er in dit boek van uit dat u met deze instructies kunt omgaan. Als hulp is verderop in dit hoofdstuk een instructie-overzicht opgenomen, waarbij ook enige oefenprogramma's zijn toegevoegd.



Afb. 12.5. CPU, gezien vanuit het programma.

We zullen in hoofdstuk 19 nog nader ingaan op de noodzaak, die zich soms voordoet om toch via de ASSEMBLER taal programma's in machinetaal te schrijven. De belangrijkste redenen zijn: executiesnelheid, geheugengebruik, de mogelijkheid om in (P)ROM te werken. De executiesnelheid van machinetaalprogramma's is groter omdat ze direct door de computer kunnen worden uitgevoerd. Soms is die grotere snelheid domweg nodig om snel genoeg te kunnen reageren op signalen van buitenaf. Verder is belangrijk dat de executiesnelheid van iedere instructie precies bekend is, zodat de executiesnelheid van een programmadeel precies vastligt. Het kleinere geheugengebruik van machinetaalprogramma's is soms ook nodig

voor een haalbare oplossing van een probleem. Doordat geheugen steeds goedkoper wordt, wordt deze reden, zeker in de toekomst, minder belangrijk. Tenslotte moet worden vermeld dat het gewoonlijk eenvoudig is machinetaalprogramma's permanent in (P)ROM geheugen onder te brengen, mits programma's en de bijbehorende data vanaf het begin gescheiden worden gehouden. Voor BASIC programma's is dit gewoonlijk niet mogelijk. Zij moeten bij hernieuwd gebruik van cassette (of floppy disk) worden geladen of opnieuw ingetikt. *Microprocessors 1* is vrij uitgebreid ingegaan op machinetaalprogrammering met behulp van een assembler. De assembler is dus ook een vertaler die zelf in machinetaal is geschreven en op cassette of in (P)ROM ter beschikking wordt gesteld.
Het beeld dat de programmeur van de CPU heeft is in afbeelding 12.5 weergegeven.

12.5. De 8080 instructies

Er volgen nu een aantal tabellen waarin de 78 instructies, die de 8080 microprocessor kent, zijn samengevat. Ze zijn verdeeld in *vijf* groepen naar hun functionele samenhang. Een indeling die aansluit op het eerder in dit hoofdstuk behandelde.
De logische en rekenkundige instructies zijn apart gegroepeerd.
De notatie die in de tabellen wordt gehanteerd, vraagt enige toelichting.

Symbolen en tekens
Om te beginnen wordt in Tabel A1 (p. 225) aangegeven wat de betekenis is van de gebruikte symbolen en tekens. Voor het 'lezen' van de notatie in de volgende tabellen is Tabel A1 onontbeerlijk.

Operatiebeschrijving
Van elke instructie wordt in Tabel A2 (p. 226 e.v.) *derde* kolom een korte beschrijving gegeven van de werking. Dit gebeurt in termen van 'toewijzing' (Engels: 'assignment').

– De instructie LDA (Tabel A2, regel 1) wordt als volgt beschreven:

  $(A) \leftarrow ((\text{byte } 3)\ (\text{byte } 2))$

  Dit betekent: de inhoud van register A (accumulator) wordt gelijk gemaakt aan de inhoud van de geheugenplaats, waarvan het adres in de bytes 2 en 3 van de instructie staat.

– De instructie MOV r1, r2, (Tabel A2, regel 11) wordt als volgt beschreven:

  $(r1) \leftarrow (r2)$

  Dit betekent: de inhoud van het als eerste genoemde register wordt gelijk gemaakt aan de inhoud van het als tweede genoemde.

Instructielengte
In kolom 6 van Tabel A2 wordt aangegeven hoeveel bytes (= woorden) de instructies beslaan en dus, impliciet, hoeveel fetch-cycli er nodig zijn om de instructies over te brengen naar de microprocessor.

Tabel A1: Symbolen en tekens

Symbool/teken	Betekenis
A	register A (accumulator)
B	register B
C	register C
D	register D
E	register E
H	register H
L	register L
DDD/SSS	het bit-patroon dat een van de registers aanwijst als doel (DDD) of als oorsprong (SSS) D staat voor *d*estination, S voor *s*ource Codering:

DDD of SSS	register
111	A
000	B
001	C
010	D
011	E
100	H
101	L

Symbool/teken	Betekenis
byte 2	de tweede byte (het tweede woord) van de instructie
byte 3	de derde byte (het derde woord) van de instructie
poort	8-bits adres van input- of outputpoort
r, r1, r2	een van de registers A, B, C, D, E, H, L
PC	16-bits programmateller (program couter) PCH en PCL zijn namen voor het 'higher' en 'lower' deel van het adres
SP	16-bits stackpointer-register De namen SPH en SPL worden gebruikt voor het 'higher' en 'lower' deel van het adres
( )	'de inhoud van' het tussen haakjes genoemde
←	'wordt vervangen door' het na de pijl genoemde
∧	logische EN
⊻	exclusieve OF
∨	OF
+	optellen
–	twee-complement aftrekken
*	vermenigvuldigen
↔	verwisselen
$\overline{A}$	het een-complement van A
n	herstart-nummers 0-7
NNN	binaire weergave 000-111 voor herstartnummers 0-7
•	niet-beïnvloed
0	conditieflag niet gezet
1	conditieflag gezet
x	onbekend
↕	flags beïnvloed volgens standaards, uitgezonderd dat wat apart wordt vermeld
M	geheugenplaats (Engels: 'memory')

Tabel A2: De 8080 instructieset*

Mnemonic	Description	Operation	Op Code								No. of Bytes	No. of Machine (M) Cycles	No. of States (T)	Condition Flags				
			D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0				S	Z	AC	P	CY
DATA TRANSFER GROUP																		
LDA	Load Accumulator Direct	(A) ← ((byte 3) (byte 2))	0	0	1	1	1	0	1	0	3	4	13					
LDAX B	Load Accumulator Indirect	(A) ← ((B) (C))	0	0	0	0	1	0	1	0	1	2	7					
LDAX D	Load Accumulator Indirect	(A) ← ((D) (E))	0	0	0	1	1	0	1	0	1	2	7					
LHLD	Load H and L Direct	(L) ← ((byte 3) (byte 2)) (H) ← ((byte 3) (byte 2) +1)	0	0	1	0	1	0	1	0	3	5	16					
LXI B	Load Immediate, Registers B and C	(B) ← (byte 3) (C) ← (byte 2)	0	0	0	0	0	0	0	1	3	3	10					
LXI D	Load Immediate, Registers D and E	(D) ← (byte 3) (E) ← (byte 2)	0	0	0	1	0	0	0	1	3	3	10					
LXI H	Load Immediate, Registers H and L	(H) ← (byte 3) (L) ← (byte 2)	0	0	1	0	0	0	0	1	3	3	10					
LXI SP	Load Immediate, Stack Pointer	(SPH) ← (byte 3) (SPL) ← (byte 2)	0	0	1	1	0	0	0	1	3	3	10	(Flags Not Affected)				
MOV M, r	Move to Memory	((H) (L)) ← (r)	0	1	1	1	0	*S*	*S*	*S*	1	2	7					
MOV r, M	Move from Memory	(r) ← ((H) (L))	0	1	*D*	*D*	*D*	1	1	0	1	2	7					
MOV r1, r2	Move Registers	(r1) ← (r2)	0	1	*D*	*D*	*D*	*S*	*S*	*S*	1	1	5					
MVI M	Move to Memory Immediate	((H) (L)) ← (byte 2)	0	0	1	1	0	1	1	0	2	3	10					
MVI r	Move Immediate	(r) ← (byte 2)	0	0	*D*	*D*	*D*	1	1	0	2	2	7					
SHLD	Store H and L Direct	((byte 3) (byte 2)) ← (L) ((byte 3) (byte 2) + 1) ← (H)	0	0	1	0	0	0	1	0	3	5	16					
STA	Store Accumulator Direct	((byte 3) (byte 2)) ← (A)	0	0	1	1	0	0	1	0	3	4	13					
STAX B	Store Accumulator Indirect	((B) (C)) ← (A)	0	0	0	0	0	0	1	0	1	2	7					
STAX D	Store Accumulator Indirect	((D) (E)) ← (A)	0	0	0	1	0	0	1	0	1	2	7					
XCHG	Exchange H and L with D and E	(H) ↔ (D) (L) ↔ (E)	1	1	1	0	1	0	1	1	1	1	4					
ARITHMETIC GROUP																		
ACI	Add Immediate with Carry	(A) ← (A) + (byte 2) + (CY)	1	1	0	0	1	1	1	0	2	2	7	↕	↕	↕	↕	↕
ADC M	Add Memory with Carry	(A) ← (A) + ((H) (L)) + (CY)	1	0	0	0	1	1	1	0	1	2	7	↕	↕	↕	↕	↕
ADC r	Add Register with Carry	(A) ← (A) + (r) + (CY)	1	0	0	0	1	*S*	*S*	*S*	1	1	4	↕	↕	↕	↕	↕
ADD M	Add Memory	(A) ← (A) + ((H) (L))	1	0	0	0	0	1	1	0	1	2	7	↕	↕	↕	↕	↕
ADD r	Add Register	(A) ← (A) +(r)	1	0	0	0	0	*S*	*S*	*S*	1	1	4	↕	↕	↕	↕	↕
ADI	Add Immediate	(A) ← (A) + (byte 2)	1	1	0	0	0	1	1	0	1	2	7	↕	↕	↕	↕	↕
DAA	Decimal Adjust Accumulator	8-bit number in Accumulator is converted to two 4-bit BCD digits	0	0	1	0	0	1	1	1	1	1	4	↕	↕	↕	↕	↕
DAD B	Add B and C to H and L	(H) (L) ← (H) (L) + (B) (C)	0	0	0	0	1	0	0	1	1	3	10	·	·	·	·	↕
DAD D	Add D and E to H and L	(H) (L) ← (H) (L) + (D) (E)	0	0	0	1	1	0	0	1	1	3	10	·	·	·	·	↕
DAD H	Add H and L to H and L	(H) (L) ← (H) (L) + (H) (L)	0	0	1	0	1	0	0	1	1	3	10	·	·	·	·	↕
DAD SP	Add Stack Pointer to H and L	(H) (L) ← (H) (L) + (SP)	0	0	1	1	1	0	0	1	1	3	10	·	·	·	·	↕
DCR M	Decrement Memory	((H) (L)) ← ((H) (L)) - 1	0	0	1	1	0	1	0	1	1	3	10	↕	↕	↕	↕	·
DCR r	Decrement Register	(r) ← (r) - 1	0	0	*D*	*D*	*D*	1	0	1	1	1	5	↕	↕	↕	↕	·
DCX B	Decrement Registers B and C	(B) (C) ← (B) (C) - 1	0	0	0	0	1	0	1	1	1	1	5	·	·	·	·	·
DCX D	Decrement Registers D and E	(D) (E) ← (D) (E) - 1	0	0	0	1	1	0	1	1	1	1	5	·	·	·	·	·
DCX H	Decrement Registers H and L	(H) (L) ← (H) (L) - 1	0	0	1	0	1	0	1	1	1	1	5	·	·	·	·	·
DCX SP	Decrement Stack Pointer	(SP) ← (SP) - 1	0	0	1	1	1	0	1	1	1	1	5	·	·	·	·	·
INR M	Increment Memory	((H) (L)) ← ((H) (L)) + 1	0	0	1	1	0	1	0	0	1	3	10	↕	↕	↕	↕	·
INR r	Increment Register	(r) ← (r) + 1	0	0	*D*	*D*	*D*	1	0	0	1	1	5	↕	↕	↕	↕	·
INX B	Increment Registers B and C	(B) (C) ← (B) (C) + 1	0	0	0	0	0	0	1	1	1	1	5	·	·	·	·	·
INX D	Increment Registers D and E	(D) (E) ← (D) (E) +1	0	0	0	1	0	0	1	1	1	1	5	·	·	·	·	·
INX H	Increment Registers H and L	(H) (L) ← (H) (L) + 1	0	0	1	0	0	0	1	1	1	1	5	·	·	·	·	·

* Deze tabel is met toestemming overgenomen uit *Microprocessor System Design Manual.*

Tabel A2: De 8080 instructieset (vervolg)

Mnemonic	Description	Operation	Op Code D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	No. of Bytes	No. of Machine (M) Cycles	No. of States (T)	Condition Flags S	Z	AC	P	CY
ARITHMETIC GROUP (Continued)																		
INX SP	Increment Stack Pointer	(SP) ← (SP) + 1	0	0	1	1	0	0	1	1	1	1	5	·	·	·	·	·
SBB M	Subtract Memory with Borrow	(A) ← (A) - ((H) (L)) - (CY)	1	0	0	1	1	1	1	0	1	2	7	‡	‡	‡	‡	‡
SBB r	Subtract Register with Borrow	(A) ← (A) - (r) - (CY)	1	0	0	1	1	S	S	S	1	1	4	‡	‡	‡	‡	‡
SBI	Subtract Immediate with Borrow	(A) ← (A) - (byte 2) - (CY)	1	1	0	1	1	1	1	0	2	2	7	‡	‡	‡	‡	‡
SUB M	Subtract Memory	(A) ← (A) - ((H) (L))	1	0	0	1	0	1	1	0	1	2	7	‡	‡	‡	‡	‡
SUB r	Subtract Register	(A) ← (A) - (r)	1	0	0	1	0	S	S	S	1	1	4	‡	‡	‡	‡	‡
SUI	Subtract Immediate	(A) ← (A) - (byte 2)	1	1	0	1	0	1	1	0	2	2	7	‡	‡	‡	‡	‡
LOGICAL GROUP																		
ANA M	AND Memory	(A) ← (A) ∧ ((H) (L))	1	0	1	0	0	1	1	0	1	2	7	‡	‡	‡[d]	‡	0
ANA r	AND Register	(A) ← (A) ∧ (r)	1	0	1	0	0	S	S	S	1	1	4	‡	‡	‡[d]	‡	0
ANI	AND Immediate	(A) ← (A) ∧ (byte 2)	1	1	1	0	0	1	1	0	2	2	7	‡	‡	‡[d]	‡	0
CMA	Complement Accumulator	(A) ← $(\overline{A})$	0	0	1	0	1	1	1	1	1	1	4	·	·	·	·	·
CMC	Complement Carry	(CY) ← $(\overline{CY})$	0	0	1	1	1	1	1	1	1	1	4	·	·	·	·	‡
CMP M	Compare Memory	(A) – ((H) (L))	1	0	1	1	1	1	1	0	1	2	7	‡	‡[a]	‡	‡	‡[a]
CMP r	Compare Register	(A) – (r)	1	0	1	1	1	S	S	S	1	1	4	‡	‡[b]	‡	‡	‡[b]
CPI	Compare Immediate	(A) – (byte 2)	1	1	1	1	1	1	1	0	2	2	7	‡	‡[c]	‡	‡	‡[c]
ORA M	OR Memory	(A) ← (A) V ((H) (L))	1	0	1	1	0	1	1	0	1	2	7	‡	‡	0	‡	0
ORA r	OR Register	(A) ← (A) V (r)	1	0	1	1	0	S	S	S	1	1	4	‡	‡	0	‡	0
ORI	OR Immediate	(A) ← (A) V (byte 2)	1	1	1	1	0	1	1	0	2	2	7	‡	‡	0	‡	0
RAL	Rotate Left through Carry	$(A_{n+1}) \leftarrow (A_n)$; $(CY) \leftarrow (A_7)$ $(A_0) \leftarrow (CY)$	0	0	0	1	0	1	1	1	1	1	4	·	·	·	·	‡
RAR	Rotate Right through Carry	$(A_n) \leftarrow (A_{n+1})$; $(CY) \leftarrow (A_0)$ $(A_7) \leftarrow (CY)$	0	0	0	1	1	1	1	1	1	1	4	·	·	·	·	‡
RLC	Rotate Left	$(A_{n+1}) \leftarrow (A_n)$; $(A_0) \leftarrow (A_7)$ $(CY) \leftarrow (A_7)$	0	0	0	0	0	1	1	1	1	1	4	·	·	·	·	‡
RRC R	Rotate Right	$(A_n) \leftarrow (A_{n-1})$; $(A_7) \leftarrow (A_0)$ $(CY) \leftarrow (A_0)$	0	0	0	0	1	1	1	1	1	1	4	·	·	·	·	‡
STC	Set Carry	(CY) ← 1	0	0	1	1	0	1	1	1	1	1	4	·	·	·	·	1
XRA M	Exclusive OR Memory	(A) ← (A) ∀ ((H) (L))	1	0	1	0	1	1	1	0	1	2	7	‡	‡	0	‡	0
XRA r	Exclusive OR Register	(A) ← (A) ∀ (r)	1	0	1	0	1	S	S	S	1	1	4	‡	‡	0	‡	0
XRI	Exclusive OR Immediate	(A) ← (A) ∀ (byte 2)	1	1	1	0	1	1	1	0	2	2	7	‡	‡	0	‡	0
BRANCH GROUP																		
CALL	Call Unconditional	((SP) - 1) ← (PCH) ((SP) - 2) ← (PCL) (SP) ← (SP) - 2 (PC) ← (byte 3) (byte 2)	1	1	0	0	1	1	0	1	3	5	17	(Flags Not Affected)				
CC	Call on Carry	If CY = 1, ((SP) - 1) ← (PCH) ((SP) - 2) ← (PCL) (SP) ← (SP) - 2 (PC) ← (byte 3) (byte 2)	1	1	0	1	1	1	0	0	3	3/5	11/17					
CM	Call on Minus	If S = 1, ((SP) - 1) ← (PCH) ((SP) - 2) ← (PCL) (SP) ← (SP) - 2 (PC) ← (byte 3 (byte 2)	1	1	1	1	1	1	0	0	3	3/5	11/17					

Notes: a. Z = 1 if (A) = (H) (L); CY = 1 if (A) < (H) (L) b. Z = 1 if (A) = (r); CY = 1 if (A) < (r) c. Z = 1 if (A) = (byte 2); CY = 1 if (A) < (byte 2) d. Treated as if an arithmetic operation were performed

Tabel A2: De 8080 instructieset (vervolg)

Mnemonic	Description	Operation	Op Code								No. of Bytes	No. of Machine (M) Cycles	No. of States (T)	Condition Flags				
			D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0				S	Z	AC	P	CY
BRANCH GROUP (Continued)																		
CNC	Call on No Carry	If CY = 0, ((SP) - 1) ← (PCH) ((SP) - 2) ← (PCL) (SP) ← (SP) - 2 (PC) ← (byte 3) (byte 2)	1	1	0	1	0	1	0	0	3	3/5	11/17					
CNZ	Call on Not Zero	If Z = 0, ((SP) - 1) ← (PCH) ((SP) - 2) ← (PCL) (SP) ← (SP) - 2 (PC) ← (byte 3) (byte 2)	1	1	0	0	0	1	0	0	3	3/5	11/17	(Flags Not Affected)				
CP	Call on Positive	If S = 0, ((SP) - 1) ← (PCH) ((SP) - 2) ← (PCL) (SP) ← (SP) - 2 (PC) ← (byte 3) (byte 2)	1	1	1	1	0	1	0	0	3	3/5	11/17					
CPE	Call on Parity Even	If P = 1, ((SP) - 1) ← (PCH) ((SP) - 2) ← (PCL) (SP) ← (SP) - 2 (PC) ← (byte 3) (byte 2)	1	1	1	0	1	1	0	0	3	3/5	11/17					
CPO	Call on Parity Odd	If P = 0, ((SP) - 1) ← (PCH) ((SP) - 2) ← (PCL) (SP) ← (SP) - 2 (PC) ← (byte 3) (byte 2)	1	1	1	0	0	1	0	0	3	3/5	11/17					
CZ	Call on Zero	If Z = 1, ((SP) - 1) ← (PCH) ((SP) - 2) ← (PCL) (SP) ← (SP) - 2 (PC) ← (byte 3) (byte 2)	1	1	0	0	1	1	0	0	3	3/5	11/17					
JC	Jump on Carry	If CY = 1, (PC) ← (byte 3) (byte 2)	1	1	0	1	1	0	1	0	3	3	10					
JM	Jump on Minus	If S = 1, (PC) ← (byte 3) (byte 2)	1	1	1	1	1	0	1	0	3	3	10					
JMP	Jump Unconditional	(PC) ← (byte 3) (byte 2)	1	1	0	0	0	0	1	1	3	3	10					
JNC	Jump on No Carry	If CY = 0, (PC) ← (byte 3) (byte 2)	1	1	0	1	0	0	1	0	3	3	10					
JNZ	Jump on Not Zero	If Z = 0, (PC) ← (byte 3) (byte 2)	1	1	0	0	0	0	1	0	3	3	10					
JP	Jump on Positive	If S = 0, (PC) ← (byte 3) (byte 2)	1	1	1	1	0	0	1	0	3	3	10					
JPE	Jump on Parity Even	If P = 1, (PC) ← (byte 3) (byte 2)	1	1	1	0	1	0	1	0	3	3	10					
JPO	Jump on Parity Odd	If P = 0, (PC) ← (byte 3) (byte 2)	1	1	1	0	0	0	1	0	3	3	10					
JZ	Jump on Zero	If Z = 1, (PC) ← (byte 3) (byte 2)	1	1	0	0	1	0	1	0	3	3	10					
PCHL	H and L to Program Counter	(PCH) ← (H) (PCL) ← (L)	1	1	1	0	1	0	0	1	1	1	5					
RC	Return on Carry	If CY = 1, (PCL) ← ((SP)) (PCH) ← ((SP) + 1) (SP) ← (SP) + 2	1	1	0	1	1	0	0	0	1	1/3	5/11					

Tabel A2: De 8080 instructieset (vervolg)

Mnemonic	Description	Operation	Op Code D7	D6	D5	D4	D3	D2	D1	D0	No. of Bytes	No. of Machine (M) Cycles	No. of States (T)	Condition Flags S	Z	AC	P	CY
BRANCH GROUP (Continued)																		
RET	Return	(PCL) ← ((SP)); (PCH) ← ((SP) + 1); (SP) ← (SP) + 2;	1	1	0	0	1	0	0	1	1	3	10					
RM	Return on Minus	If S = 1, (PCL) ← ((SP)) (PCH) ← ((SP) + 1) (SP) ← (SP) + 2	1	1	1	1	1	0	0	0	1	1/3	5/11					
RNC	Return on No Carry	If CY = 0, (PCL) ← ((SP)) (PCH) ← ((SP) + 1) (SP) ← (SP) + 2	1	1	0	1	0	0	0	0	1	1/3	5/11					
RNZ	Return on Not Zero	If Z = 0, (PCL) ← ((SP)) (PCH) ← ((SP) + 1) (SP) ← (SP) + 2	1	1	0	0	0	0	0	0	1	1/3	5/11					
RP	Return on Positive	If S = 0, (PCL) ← ((SP)) (PCH) ← ((SP) + 1) (SP) ← (SP) + 2	1	1	1	1	0	0	0	0	1	1/3	5/11					
RPE	Return on Parity Even	If P = 1, (PCL) ← ((SP)) (PCH) ← ((SP) + 1) (SP) ← (SP) + 2	1	1	1	0	1	0	0	0	1	1/3	5/11					
RPO	Return on Parity Odd	If P = 0, (PCL) ← ((SP)) (PCH) ← ((SP) + 1) (SP) ← (SP) + 2	1	1	1	0	0	0	0	0	1	1/3	5/11					
RST	Restart	((SP) - 1) ← (PCH) ((SP) - 2) ← (PCL) (SP) ← (SP) - 2 (PC) ← 8 * (NNN)	1	1	*N*	*N*	*N*	1	1	1	1	3	11					
RZ	Return on Zero	If Z = 1, (PCL) ← ((SP)) (PCH) ← ((SP) + 1) (SP) ← (SP) + 2	1	1	0	0	1	0	0	0	1	1/3	5/11					
STACK, I/O, AND MACHINE CONTROL GROUP																		
DI	Disable Interrupts	The interrupt system is disabled following the execution of the DI instruction.	1	1	1	1	0	0	1	1	1	1	4	•	•	•	•	•
EI	Enable Interrupts	The interrupt system is enabled following the execution of next instruction.	1	1	1	1	1	0	1	1	1	1	4	•	•	•	•	•
HLT	Halt	Processor is stopped, registers and flags are unaffected.	0	1	1	1	0	1	1	0	1	1	7	•	•	•	•	•
IN	Input	(A) ← (data)	1	1	0	1	1	0	1	1	2	3	10	•	•	•	•	•
NOP	No Operation	No operation is performed, registers and flags are unaffected.	0	0	0	0	0	0	0	0	1	1	4	•	•	•	•	•
OUT	Output	(data) ← (A)	1	1	0	1	0	0	1	1	2	3	10	•	•	•	•	•
POP B	Pop Registers B and C off Stack	(C) ← ((SP)) (B) ← ((SP) + 1) (SP) ← (SP) + 2)	1	1	0	0	0	0	0	1	1	3	10	•	•	•	•	•

Tabel A2: De 8080 instructieset (vervolg)

Mnemonic	Description	Operation	Op Code								No. of Bytes	No. of Machine (M) Cycles	No. of States (T)	Condition Flags				
			D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0				S	Z	AC	P	CY
STACK, I/O, AND MACHINE CONTROL GROUP (Continued)																		
POP D	Pop Registers D and E off Stack	(E) ← ((SP)) (D) ← ((SP) + 1) (SP) ← (SP) + 2	1	1	0	1	0	0	0	1	1	3	10	•	•	•	•	•
POP H	Pop Registers H and L off Stack	(L) ← ((SP)) (H) ← ((SP) + 1) (SP) ← (SP) + 2	1	1	1	0	0	0	0	1	1	3	10	•	•	•	•	•
POP PSW	Pop Accumulator and Flags off Stack	(CY) ← $((SP))_0$ (P) ← $((SP))_2$ (AC) ← $((SP))_4$ (Z) ← $((SP))_6$ (S) ← $((SP))_7$ (A) ← ((SP) + 1) (SP) ← (SP) + 2	1	1	1	1	0	0	0	1	1	3	10	↕	↕	↕	↕	↕
PUSH B	Push Registers B and C on Stack	((SP) - 1) ← (B) ((SP) - 2) ← (C) (SP) ← (SP) - 2	1	1	0	0	0	1	0	1	1	3	1	•	•	•	•	•
PUSH D	Push Registers D and E on Stack	((SP) - 1) ← (D) ((SP) - 2) ← (E) (SP) ← (SP) - 2	1	1	0	1	0	1	0	1	1	3	11	•	•	•	•	•
PUSH H	Push Registers H and L on Stack	((SP) - 1) ← (H) ((SP) - 2) ← (L) (SP) ← (SP) - 2	1	1	1	0	0	1	0	1	1	3	11	•	•	•	•	•
PUSH PSW	Push Accumulator and Flags on Stack	((SP) - 1) ← (A) $((SP) - 2)_0$ ← (CY) $((SP) - 2)_1$ ← 1 $((SPf - 2)_2$ ← (P) $((SP) - 2)_3$ ← 0 $((SP) - 2)_4$ ← (AC) $((SP) - 2)_5$ ← 0 $((SP) - 2)_6$ ← (Z) $((SP) - 2)_7$ ← (S) (SP) ← (SP) - 2	1	1	1	1	0	1	0	1	1	3	11	•	•	•	•	•
SPHL	Move H and L to Stack Pointer	(SP) ← (H) (L)	1	1	1	1	1	0	0	1	1	1	5	•	•	•	•	•
XTHL	Exchange Top of Stack with H and L	(L) ↔ ((SP)) (H) ↔ ((SP) + 1)	1	1	1	0	0	0	1	1	1	5	18	•	•	•	•	•

Enkele uitwerkingen van Tabel A2

Mnemonic	Description	Operation	Op Code D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	No. of Bytes	No. of Machine (M) Cycles	No. of States (T)	Condition Flags S	Z	AC	P	CY
DATA TRANSFER GROUP																		
LDA	Load Accumulator Direct	(A) ← ((byte 3) (byte 2))	0	0	1	1	1	0	1	0	3	4	13	·	·	·	·	·

Register A krijgt de inhoud van de geheugenplaats met het adres dat in byte 3/byte 2 staat.

Operatiecode : <0011 1010> <byte 2> <byte 3> of 3A <byte 2> <byte 3>
Tijdsduur : 13 states (stappen) à 500 nanosec. = 6,5 microsec.

Mnemonic	Description	Operation	Op Code D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	No. of Bytes	No. of Machine (M) Cycles	No. of States (T)	Condition Flags S	Z	AC	P	CY
DATA TRANSFER GROUP																		
MOV M, r	Move to Memory	((H) (L)) ← (r)	0	1	1	1	0	S	S	S	1	2	7	·	·	·	·	·

Inhoud van source-register r (aangegeven door bits SSS) gaat naar geheugenplaats, aangewezen door de inhoud van registerpaar HL.

Operatiecode : 0111 0SSS
bij : MOV M,A ; SSS = 111 : operatiecode is 0111 0111 = 77 hexadecimaal.
Tijdsduur : 7 states (stappen) à 500 nanosec. = 3,5 microsec.

Mnemonic	Description	Operation	Op Code D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	No. of Bytes	No. of Machine (M) Cycles	No. of States (T)	Condition Flags S	Z	AC	P	CY
ARITHMETIC GROUP																		
ADD	Add Register	(A) ← (A) + (r)	1	0	0	0	0	S	S	S	1	1	4	↕	↕	↕	↕	↕

Inhoud accumulator wordt vermeerderd met de inhoud source-register aangegeven door SSS. Alle flags worden na executie aangepast aan de waarde van het resultaat.

Operatiecode : 1000 0SSS
bij : ADD C; SSS = 001; operatiecode is 1000 0001 = 81 hexadecimaal.
Tijdsduur : 4 states (stappen) à 500 nanosec. = 2 microsec.

Machinecycli en klokcycli
In kolom 7 van Tabel A2 staat hoeveel machinecycli de uitvoering van een instructie vergt, c.q. het aantal malen dat er ten behoeve van de instructie van de databus gebruik wordt gemaakt.
In kolom 8 wordt vermeld hoeveel klokcycli er met de instructie gemoeid zijn tijdens de uitvoering.

Conditieflags
Kolom 9 van Tabel A2 geeft tenslotte aan welke relatie de instructie heeft met de conditieflags.
Er zijn vijf conditieflags betrokken bij de uitvoering van een instructie:

ZERO (Z)	: wordt gezet als het resultaat van de instructie de waarde 0 heeft;
SIGN (S)	: wordt gezet als het meest significante bit (c.q. het meeste linkse) van het resultaat van de instructie een 1 is (negatief);
PARITY (P)	: wordt gezet in geval van 'even parity', dat wil zeggen als het aantal enen in het instructie-resultaat even is;
CARRY (CY)	: wordt gezet als de instructie resulteert in een *carry* (optelling) of een *borrow* (aftrekking of vergelijking);
AUXILIARY CARRY (AC)	: wordt gezet als de instructie resulteert in een carry van bit 3 naar bit 4 van de resultaatwaarde. Optellingen, aftrekkingen, increment- en decrement-functies, vergelijkingen en logische bewerkingen over 1 byte beïnvloeden de AC-flag. Vooral is deze flag echter van belang bij de DAA-instructie en optelling en increment-instructies.

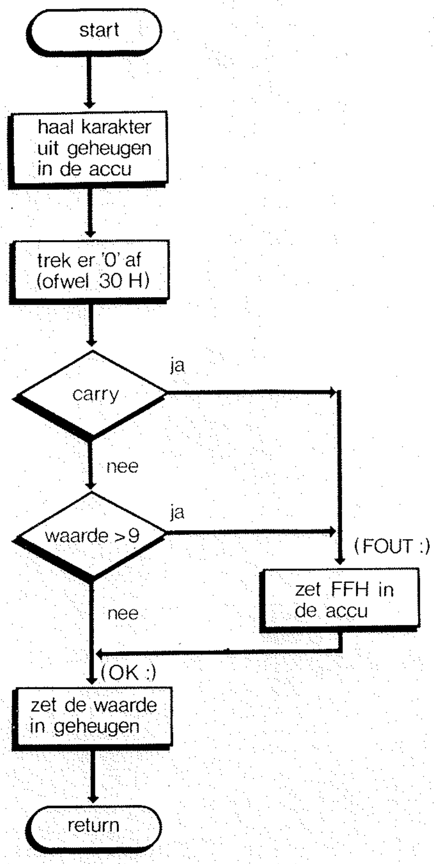
(Het 'wordt gezet' houdt in dat de betreffende flagbit op één wordt gezet. Het flagbit wordt niet gezet (krijgt dus de waarde 0 als aan de voorwaarde in kwestie niet is voldaan)).
Nu zullen we nog enige voorbeelden laten zien van veel voorkomende programmeer-probleempjes voor de ASSEMBLER programmeur zoals die zich ook in de BASIC interpreter kunnen voordoen. Daarvoor vertalen we eerst een ASCII teken, dat een decimaal getal voorstelt, naar zijn decimale waarde. We veronderstellen dat de ASCII waarde aan het begin op adres 11ØØ (hexadecimaal) staat. Na afloop van de executie staat er de decimale waarde of er staat een foutmelding in de vorm van een string met énen. De ACII waarde stelde dan geen decimaal getal voor. We maken gebruik van de wetenschap dat de getallen Ø tot en met 9 de waarde 3ØH tot en met 39H hebben (zie de ASCII-tabel in hoofdstuk 10).

Een voorbeeld van een ASSEMBLER-programma voor dit doel geeft afbeelding 12.6.

De flowchart van dit programma is weergegeven in afbeelding 12.7.
In BASIC roepen we een in machinetaal geschreven programma aan met het statement CALLM adres. CALLM staat voor CALL Machine language program ofwel roep het machinetaalprogramma op.

```
ORG   1200H        ;BEGINADRES VAN HET PROGRAMMA
LDA   1100H        ;HAAL HET KARAKTER OP
SUI   '0'          ;TREK DE ASCII-WAARDE 0 ER AF
JC    FOUT         ;ALS ER EEN CARRY KOMT IS HET
                   ;KARAKTER GEEN CIJFER.
CPI   10           ;IS HET RESULTAAT GROTER DAN 9
JNC   FOUT         ;DAN FOUTMELDING.
JMP   OK           ;ANDERS IS HET GOED
MVI   A,OFFH       ;ZET ENEN IN DE ACCU
STA   1100H        ;ZET DE ACCUWAARDE IN GEHEUGEN
RET                ;KEER TERUG NAAR AANROEPEND
                   ;PROGRAMMA.
END   START
```

Afb. 12.6.



Afb. 12.7. Flowchart ASCII naar decimaalvertaling.

Het adres is hierbij het hexadecimale beginadres van het machinetaal programma. BASIC behandelt dit programma dan als een subroutine. Om alles goed te laten verlopen moeten we dit machinetaalprogramma afsluiten met een RETURN instructie. Deze RETURN instructie is vergelijkbaar met het RET statement uit BASIC (verderop in dit hoofdstuk zullen we de 8080 machinetaal en assembleertaal instructies behandelen). Het is vaak prettig het aan te roepen adres in hexadecimale weergave te mogen geven, omdat dit gewoonlijk overeenkomt met weergave op papier (listing) van een machinetaal programma. Dan gebruiken we CALLM # hexadres. Hexadres geeft dan het adres hexadecimaal weer.
Soms wordt door de computer aan het machinetaal programma de eis gesteld dat het alle registers en flags onveranderd laat voordat met RET naar BASIC teruggegaan wordt. De programmeur dient dan eerst alle register- en flag-inhouden op te bergen. De stack is uiteraard het mechanisme bij uitstek hiervoor. Net voor de RET moeten de geborgen inhouden weer teruggezet worden.
Sommige computers staan nog een vorm van overdracht van gegevens tussen BASIC en machinetaal programma's toe als aanvulling op de in hoofdstuk 9 genoemde methodes. De aanroep van het machinetaal programma wordt dan: CALLM adres, V.
De afspraak in de Teleac/DAI personal computer hierbij is dat het register naar H,L dan het adres bevat waar de variabele V zich in het geheugen bevindt. Het machinetaal programma kan zo eenvoudig op V opereren. We zullen overigens in hoofdstuk 17 zien dat er ook een methode in BASIC is die de plaats van een variabele kan weergeven (VARPTR).
Op dit punt zijn er nogal wat verschillen tussen de computers die BASIC kennen. Sommigen staan geen aanroep van een machinetaal programma toe. Veel computers die deze aanroep wel toestaan gebruiken de naam USR in plaats van CALLM. USR staat daarbij voor USER's programma ofwel gebruikersprogramma. Ook hier weer: raadpleeg uw handleiding.
In de Teleac/DAI personal computer wordt bij gebruik van CALLM geëist dat de aanroepende routine alle 8080 registers en flags niet veranderd heeft als weer naar BASIC terug wordt gegaan. De verstandigste methode is dat de gebruiker een routine maakt die alle registers en de flags eerst op de stack opbergt. Net voor de return zullen dan deze waarden weer teruggezet moeten worden. Een moge-

```
1200            PROGR   EQU     1200H
0000                    ORG     11F0H

11F0 F5         STORE:  PUSH    PSW     ;A & FLAGS
11F1 C5                 PUSH    B       ;B & C
11F2 D5                 PUSH    D       ;D & E
11F3 E5                 PUSH    H       ;H & L
11F4 CD0012             CALL    PROGR   ;SPRING NAAR HET PROGRAMMA
11F7 E1                 POP     H       ;H & L
11F8 D1                 POP     D       ;D & E
11F9 C1                 POP     B       ;B & C
11FA F1                 POP     PSW     ;A & FLAGS
11FB C9                 RET             ;TERUG NAAR BASIC
0000                    END

NO PROGRAM ERROR(S)
```

Afb. 12.8.

```
*LIST
10    REM TOON DE ASCII NAAR DECIMAAL OMZETTING
20    INPUT "TYPE EEN WAARDE UIT 0 T/M 9 IN ";G$:PRINT
30    LET ASCII=ASC(G$)
40    POKE #1100,ASCII
50    CALLM #11F0
60    LET WAARDE%=PEEK(#1100)
70    PRINT "HET GETAL IS: ";G$
80    PRINT "DE DECIMALE WAARDE IS : ";WAARDE%
90    PRINT
100   IF WAARDE%<>255.0 GOTO 20
110   END
*RUN
TYPE EEN WAARDE UIT 0 T/M 9 IN ?0
HET GETAL IS: 0
DE DECIMALE WAARDE IS :  0

TYPE EEN WAARDE UIT 0 T/M 9 IN ?5
HET GETAL IS: 5
DE DECIMALE WAARDE IS :  5

TYPE EEN WAARDE UIT 0 T/M 9 IN ?9
HET GETAL IS: 9
DE DECIMALE WAARDE IS :  9

TYPE EEN WAARDE UIT 0 T/M 9 IN ?A
HET GETAL IS: A
DE DECIMALE WAARDE IS :  255

END PROGRAM
```

Afb. 12.9.

```
0000            START:     ORG   1200H       ;BEGINADRES VAN HET PROGRAMMA
1200 0604                  MVI   B,4         ;ZET DE BREEDTE VAN HET GETAL IN BYTES
1202 210311                LXI   H,1103H     ;BEGINADRES VAN HET EERSTE GETAL
1205 110711                LXI   D,1107H     ;BEGINADRES VAN HET TWEEDE GETAL
1208 AF                    XRA   A           ;ZET ACCU EN CARRY OP NUL
1209 1A         TELOP:     LDAX  D           ;HAAL 1 BYTE VAN HET TWEEDE GETAL
120A 8E                    ADC   M           ;TEL DIT OP BIJ HET CORRESPONDERENDE
                                             ;BYTE VAN HET EERSTE GETAL.
120B 27                    DAA               ;MAAK ER EEN DECIMAAL GETAL VAN
120C 77                    MOV   M,A         ;ZET RESULTAAT OP DE GEWENSTE PLAATS
120D 1E                    DCX   D           ;WIJS NAAR HET VOLGENDE BYTE VAN HET
                                             ;EERSTE GETAL
120E 2B                    DCX   H           ;ZO OOK VAN HET TWEEDE GETAL
120F 05                    DCR   B           ;VERLAAG NOG AF TE HANDELEN BREEDTE
1210 C20912                JNZ   TELOP       ;ALS ER NOG BYTES AF TE HANDELEN
                                             ;ZIJN OPNIEUW OPTELLEN
1213 C9                    RET               ;KEER NAAR AANROEPEND PROGRAMMA TERUG
0000                       END   START

NO PROGRAM ERROR(S)
```

Afb. 12.10.

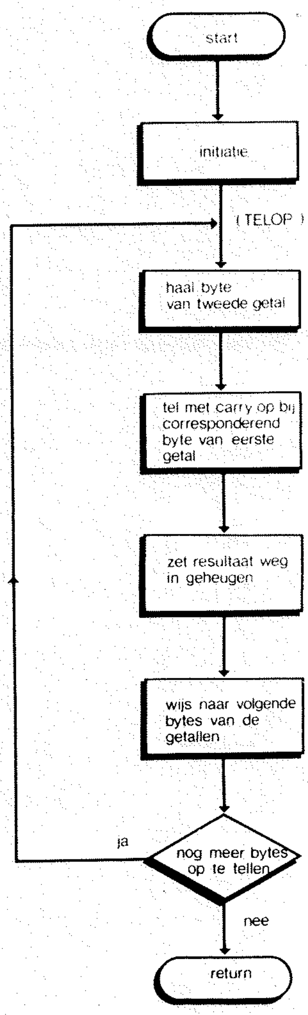
lijke routine die het opbergen verzorgt en na aanroep van machinetaal routine ook voor terugzetten zorgdraagt, wordt gegeven in afbeelding 12.8.

Dus: wanneer deze opbergroutine aan het begin van ieder machinetaal programma wordt gebruikt en de terugkeerroutine net voor de return, dan zullen er geen problemen ontstaan. De gebruiker heeft uiteraard ook de mogelijkheid deze routines als subroutine aan te roepen (ieder weer af te sluiten met RET).
Het zou fraaier zijn geweest wanneer BASIC dit opbergen en weer terughalen zelf al in CALLM had verzorgd. In sommige machines is dit ten aanzien van USR dan ook gedaan. Om te laten zien hoe het programma in BASIC kan worden aangeroepen gebruiken we het voorbeeld van afbeelding 12.9.

Van tevoren is het machinetaalprogramma in het systeem op de juiste plaats gebracht. Voor een kort programma gebeurt dat door intypen in hexadecimale vorm met een van de monitorcommando's. Nog prettiger is een laadprogramma dat ons programma meteen van de cassette laadt. De vertaling uit de symbolische weergave in mnemonics kan met de hand gebeuren. Het is uiteraard beter daarvoor een assembler te gebruiken, zoals in *Microprocessors 1* uitgebreid is behandeld.

Nu zullen we zien hoe we over meerdere bytes breedte kunnen optellen, waarbij we duidelijk gebruik maken van de carry. We stellen dat de breedte 4 bytes is. Het ene op te tellen getal staat in adres 1100 (hex.), het andere op adres 1103 (hex). Het resultaat vervangt het 4 bytes brede getal op adres 1100 (hex). Het minst significante byte staat rechts. We geven nu de listing van de assembler weer in afbeelding 12.10.

De flowchart is weergegeven in afbeelding 12.11.



Afb. 12.11. Flowchartoptelling over meerdere bytes.

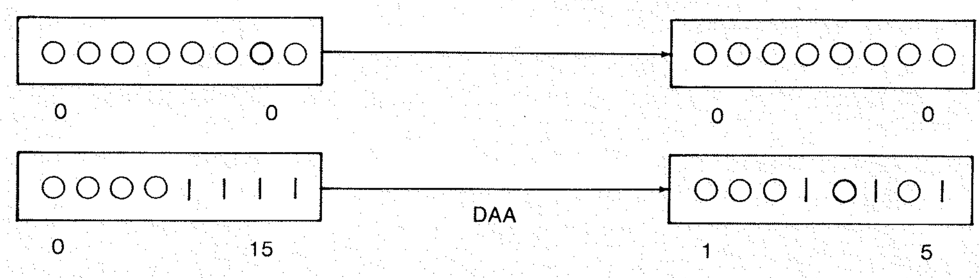
Wanneer we te maken zouden hebben met BCD (Binary Coded Decimal) getallen eveneens 4 bytes breed (8 BCD getallen), zou de routine er uit kunnen zien als afbeelding 12.12.

```
0000            START:   ORG  1200H    ;BEGINADRES VAN HET PROGRAMMA
1200 0604                MVI  B,4      ;ZET DE BREEDTE VAN HET GETAL IN BYTES
1202 210311              LXI  H,1103H  ;BEGINADRES VAN HET EERSTE GETAL
1205 110711              LXI  D,1107H  ;BEGINADRES VAN HET TWEEDE GETAL
1208 AF                  XRA  A        ;ZET ACCU EN CARRY OP NUL
1209 1A         TELOP:   LDAX D        ;HAAL 1 BYTE VAN HET TWEEDE GETAL
120A 8E                  ADC  M        ;TEL DIT OP BIJ HET CORRESPONDERENDE
                                       ;BYTE VAN HET EERSTE GETAL.
120B 77                  MOV  M,A      ;ZET RESULTAAT OP DE GEWENSTE PLAATS
120C 1B                  DCX  D        ;WIJS NAAR HET VOLGENDE BYTE VAN HET
                                       ;EERSTE GETAL.
120D 2B                  DCX  H        ;ZO OOK VAN HET TWEEDE GETAL
120E 05                  DCR; B        ;VERLAAG NOG AF TE HANDELEN BREEDTE
120F C20912              JNZ  TELOP    ;ALS ER NOG BYTES AF TE HANDELEN
                                       ;ZIJN OPNIEUW OPTELLEN.
1212 C9                  RET           ;KEER NAAR AANROEPEND PROGRAMMA TERUG
0000                     END  START

NO PROGRAM ERROR(S)
```

Afb. 12.12.

De enige afwijking ligt in het gebruik van de DAA instructie, die een binair getal (8 bits) omzet in 2 BCD getallen van ieder 4 bits. Deze instructie maakt inwendig gebruik van de AUXILARY CARRY (AC), die als FLAG in de 8080 microprocessor aanwezig is.
Er passen 2 BCD getallen in 1 byte. Ieder van die BCD getallen mag echter niet groter zijn dan 9. Ontstaat er tijdens het rekenen op een BCD plaats een getal groter dan 9 (dus een getal van 10 tot en met 15), dan wordt dit door de AUXILARY CARRY aangegeven. De DAA instructie zet dit getal weer om in 2 correcte BCD getallen, zoals afbeelding 12.13 weergeeft.



Afb. 12.13. Decimaal adjust instructie

De DAA instructie mag in de 8080 alleen na een optelling worden gebruikt om een correct resultaat te krijgen.
Sommige computers bevatten in plaats van een BASIC interpreter een *BASIC compiler*. De compiler vertaalt eerst het hele programma en geeft daarbij dan eventuele foutmeldingen tegen het taalgebruik in BASIC. Dit is vergelijkbaar met de assembler, die programma's in ASSEMBLER taal naar machinetaal vertaalt. Het voordeel van een

compiler is dat de uiteindelijke executie van het vertaalde programma sneller gaat, omdat niet meer tijdens de executie steeds regel voor regel moet worden geïnterpreteerd. Compilers zijn gewoonlijk ingewikkelder wat opbouw betreft en vereisen veel meer geheugenruimte. Daarom zijn op personal computers compilers voor BASIC nog zeldzaam.

Samenvatting hoofdstuk 12

In dit hoofdstuk hebben we gezien hoe een BASIC interpreter in principe werkt. Vervolgens hebben we een overzicht gegeven van de ASSEMBLER instructies van de 8080 microprocessor en de overwegingen voor het gebruik van ASSEMBLER programma's. Tenslotte volgden enige voorbeelden van het gebruik van ASSEMBLER in instructies.

Multiple choice-vragen bij hoofdstuk 12

A. **a.** Een BASIC interpreter gebruikt alleen ROM geheugen en geen RAM geheugen.
b. Een memory map geeft de inhoud van de geheugenplaatsen weer.

1 uitspraak a is goed en uitspraak b is goed
2 uitspraak a is fout en uitspraak b is goed
3 uitspraak a is goed en uitspraak b is fout
4 uitspraak a is fout en uitspraak b is fout

B. Een machinetaal programma, dat in BASIC wordt aangeroepen, moet om correct terug te keren naar het aanroepende BASIC-programma, afgesloten zijn met:

1 een END statement
2 een RET statement
3 een RET statement gevolgd door een END statement
4 er zijn hier geen eisen

C. **a.** De DAA instructie telt twee BCD getallen bij elkaar op.
b. De ADC M instructie telt alleen de inhoud van de geheugenplaats aangewezen door H, L op bij de inhoud van de accumulator.

1 uitspraak a is goed en uitspraak b is goed
2 uitspraak a is fout en uitspraak b is goed
3 uitspraak a is goed en uitspraak b is fout
4 uitspraak a is fout en uitspraak b is fout

D. De notatie (A) $\leftarrow$ (A) + ((H) (L)) + (CY) wordt gebruikt om het effect aan te geven van de instructie:

1 ADD M
2 ACI
3 SBB M
4 ADC M

E. Hoeveel states duur bij de 8080 de instructie MOV M, r?

1 7
2 2
3 1
4 dit hangt van r af

F. Wat is de hexadecimale code bij de 8080 voor de instructie MOV M, r met voor r het register B?

1 77
2 4E
3 70
4 36

G. **a.** Bij uitvoering van de instructie SBB M kunnen alle flags veranderen.
b. Bij de instructie ANA M wordt de carry-flag niet beïnvloed.

1 uitspraak a is goed en uitspraak b is goed
2 uitspraak a is fout en uitspraak b is goed
3 uitspraak a is goed en uitspraak b is fout
4 uitspraak a is fout en uitspraak b is fout

H. Welke 8080 instructie voert de volgende operatie uit?
Ervoor is de inhoud van A: 1 1 1 1 0 0 0 1
Erna is de inhoud van A: 0 0 0 0 0 0 0 0

1 ADI 00001110
2 XRA A
3 LDA 11110001
4 ACI 00001111

I. **a.** Gebruik van een compiler zal gewoonlijk resulteren in snellere executie van een gebruiksprogramma dan bij gebruik van een interpreter.
b. Een compiler geeft geen foutmelding tijdens de executiefase van een gebruikersprogramma.

1 uitspraak a is goed en uitspraak b is goed
2 uitspraak a is fout en uitspraak b is goed
3 uitspraak a is goed en uitspraak b is fout
4 uitspraak a is fout en uitspraak b is fout

J. De CALL instructie van de 8080 bestaat uit:

1 1 byte
2 2 bytes
3 3 bytes
4 5 bytes

Hoofdstuk 13

De microprocessor (8080)

In dit hoofdstuk gaan we verder in op wat we de architectuur van de microcomputer noemen. Dit is het samenspel van processor, geheugen en input/output via een databus. Dit samenspel wordt door besturingssignalen gecontroleerd. Voorts wordt een inzicht gegeven in de input- en outputtechnieken zoals deze bij microcomputers worden toegepast. We nemen de 8080 als uitgangspunt. Ten aanzien van andere microprocessors kunnen verschillen optreden.

13.1. De computer

De Central Processing Unit (CPU), een of meer geheugenblokken, input/outputpoorten en een bussysteem, eventueel aangevuld met een bedieningspaneel *(console)* met display en een apparaatgerichte I/O-modules, vormen de basiseenheden van een computersysteem. In *Microprocessors 1* zijn we daar uitgebreid op ingegaan. We geven hier nog eens een samenvatting.
De CPU, de centrale processor van het systeem moet de volgende functies realiseren:
— besturing;
— verwerking.

Het uitvoeren (executie) van programma-instructies wordt bestuurd door de CPU. Dat gebeurt door passende stuursignalen naar de andere eenheden te zenden en door het noodzakelijke datatransport tussen de eenheden te coördineren en te verzorgen. Alle delen van het systeem vallen onder de besturing van de CPU. Rekenkundige en logische bewerkingen worden uitgevoerd in de CPU, en wel door de Arithmetic and Logic Unit (ALU) daaruit. De bewerkingen hebben betrekking op data.
Om op zijn taak te zijn berekend, moet de CPU beschikken over de volgende onderdelen:
— programmateller;
— instructieregister en instructiedecoder;
— stackpointer;
— ALU, accumulator, registers;
— adresregister.

Programmateller = instructieteller
In de programmateller wordt bijgehouden op welk geheugenadres de eerstvolgende uit te voeren instructie staat. Nadat een instructie is uitgevoerd wordt de inhoud van de programmateller via de adresbus

naar het geheugen overgebracht. Het betreffende geheugenwoord wordt geselecteerd en de inhoud wordt gelezen. Daarna is de volgende instructie voor uitvoering beschikbaar. De programmateller wordt nu verhoogd, zodat het volgende instructie-adres bekend wordt.
Zonder programmateller kan een programma niet worden uitgevoerd omdat dan de volgorde van de instructies niet kan worden bepaald.

Instructieregister en instructiedecoder
Een uit het geheugen gelezen instructie bevat onder meer een operatiecode. Deze is bepalend voor de acties die moeten worden ondernomen. In het instructieregister wordt de instructie voor de duur van de uitvoering vastgehouden; via de instructiedecoder worden de juiste stuursignalen gegenereerd en de relevante onderdelen van de computer geactiveerd.

Stackpointer
De stackpointer is een register dat wordt gebruikt bij het uitvoeren van subroutines. Het terugkeeradres naar het onderbroken hoofdprogramma wordt hier bewaard. De stack speelt ook een belangrijke rol bij interruptverwerking. Ook dan heeft de stack een bewaarfunctie.

ALU en registers
Alle rekenkundige en logische bewerkingen vallen onder de taak van de ALU. Naast de ALU bevat de CPU:
- de accumulator voor opslag van het resultaat van rekenkundige of logische bewerkingen;
- registers waarin condities, die zich tijdens de uitvoering voordoen, kunnen worden opgeslagen, de zogenaamde flags;
- general-purpose-registers. Het gebruik van dergelijke registers — in de meeste microprocessors zijn er niet meer dan acht — maakt het mogelijk met data te werken zonder deze tussentijds steeds in het eigenlijke geheugen op te slaan. De registers vormen dus een soort tijdelijk geheugen. Er wordt tijd bespaard op het datatransport zodat met behulp van deze registers extra snel kan worden gewerkt. Meestal zijn de betreffende registers genummerd. Hun nummer is tevens hun 'adres'. Dit adres is korter dan een geheugenadres, zodoende kan ook geheugenruimte worden bespaard. De instructies zijn dan namelijk ook korter.
  Wanneer we uitgaan van een lengte van acht bits is het meestal mogelijk twee registers simultaan te gebruiken, zodat zestien bit capaciteit kan worden verkregen. Omdat de aard van het gebruik niet vaststaat spreekt men van 'general-purpose-registers'.

Adresregister
Het datatransport van en naar modules buiten de CPU wordt bestuurd via het adresregister. De programmateller en ook de stackpointer gaan via dit adresregister naar buiten. Daarnaast spreken we van het adresregister als het gaat om het lezen uit *(memory read)* of schrijven in *(memory write)* het geheugen.
Het adresregister is direct gekoppeld aan de adresbus. Men kan zelfs

zeggen dat het adresregister de adresbus is. Het gaat om een zestien bits register (2 × 8 bits). Het plaatsen van een adreswaarde in een van de registers (c.q. het 'laden' van de registers) gebeurt via de databus. Aangezien de databus is ingesteld op acht bits, zijn er twee acties nodig om een adreswaarde in de 16 bits registers te plaatsen. De beide 'helften' van de registers worden aangeduid als 'Low' (bits 0-7) en 'High' (bits 8-15). Deze aanduiding wordt gebruikt in die gevallen waarin onderscheid tussen de beide helften moet worden gemaakt. Het adresregister wordt ook gebruikt als er data van, of naar randapparatuur moet worden getransporteerd (input/output). In de instructie worden de input/outputpoorten met een code aangegeven (poortnummer). Deze code slaat dan op een van de input/outputpoorten en wordt net als een geheugenadres in het adresregister gezet.

Klok
De CPU staat volledig onder controle van een klok. Met een vaste frequentie genereert de klok signalen (pulsen). Per klokpuls kan er een elementaire handeling worden ondernomen. De klok zorgt voor orde in de opeenvolging van vele verschillende acties.

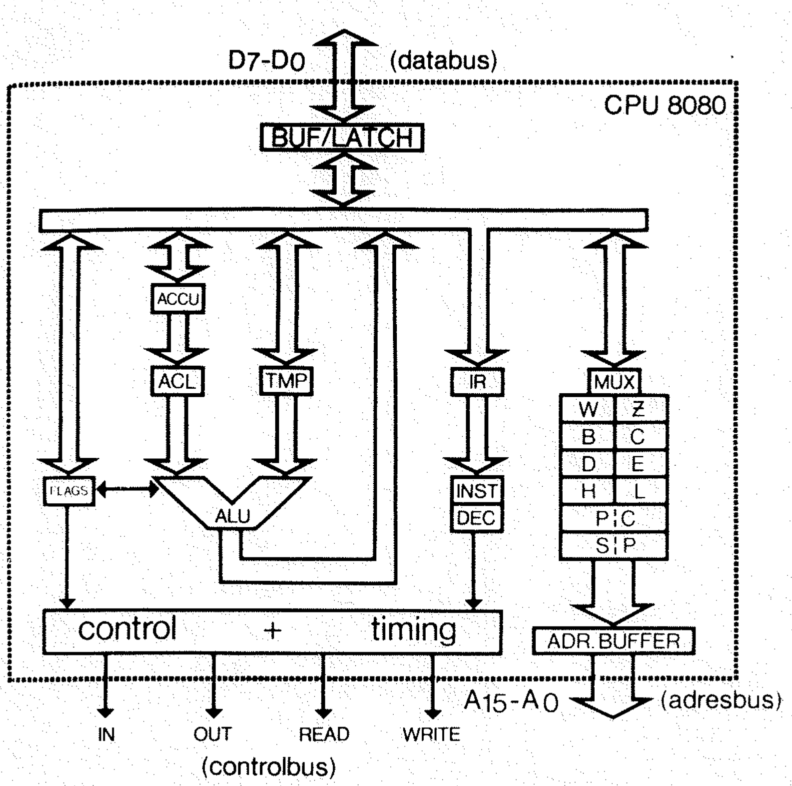
Buffers
Via input/output heeft de CPU te maken met de 'buitenwereld'. Wat binnen de eigen grenzen van de CPU gebeurt wordt in een bepaald tempo gedaan, een tempo dat zijn basis heeft in de klok. In de buffers wordt de te transporteren data tijdelijk opgeslagen. De buitenwereld werkt gewoonlijk ook in een eigen tempo. De buffers fungeren dan als een soort brievenbus voor het doorgeven van gegevens.

Stuursignalen
De CPU genereert de volgende stuursignalen:
- READ (memory): commando voor het lezen van data of instructies uit het geheugen;
- WRITE (memory): commando voor het schrijven van data in het geheugen;
- IN:.commando voor het binnenhalen van data uit een inputbuffer;
- OUT: commando voor het opslaan van data in een outputbuffer.

13.2. De 8080 CPU

Afbeelding 13.1 geeft een schematisch overzicht van een bestaande — veel toegepaste — CPU: de INTEL 8080. Wat in de voorgaande paragrafen over de CPU werd gezegd is in dit schema terug te vinden.

Het bussysteem van een microcomputer
Een bus bestaat uit een aantal, naar functie, gegroepeerde communicatielijnen. Uitgaande van de CPU onderscheidt men interne en externe bussystemen: *intern* is de bus die op de CPU-chip zelf is aangebracht om de connecties met andere componenten te bewerkstelligen; *extern* is de bus die vanuit het gezichtspunt van het totale systeem het meest van belang is. Deze bus onderhoudt de verbinding tussen de CPU, het geheugen en de I/O. Qua functie onderscheidt men drie bussystemen: de databus, de adresbus en de controlbus.

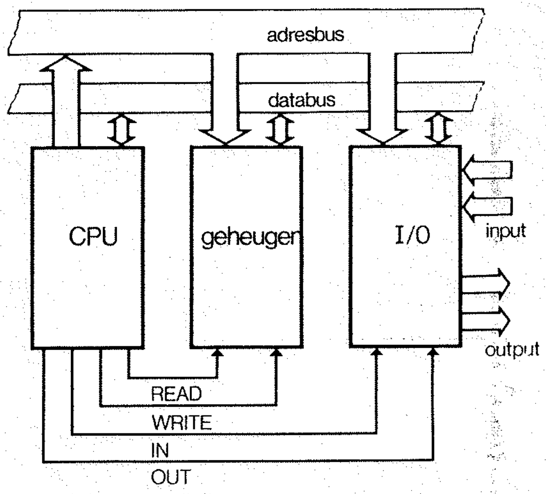


Afb. 13.1. Blokdiagram van een CPU (8080 processor)

Databus
Via de databus worden data over en weer verstuurd tussen verschillende chips van een systeem. Bijvoorbeeld vanaf een inputpoort naar de processor en vervolgens naar het geheugen. In de standaardprocessor, zoals de INTEL 8080, heeft de databus een 'breedte' van acht bits en is hij bidirectional. Dat wil zeggen: de bus kan in twee richtingen worden gebruikt, maar slechts in één richting tegelijk.

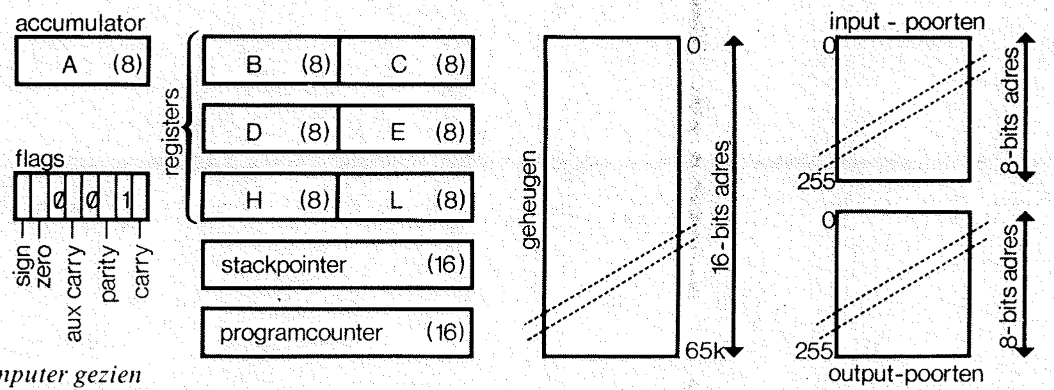
Adresbus
Het begin van de adresbus ligt bij de processor. Via deze bus worden adressen (geheugen, input/outputapparatuur, registers) overgedragen. Doorgaans betreft het een 16-bits verbinding, waarmee dus $2^{16} = 65\,566$ geheugenplaatsen kunnen worden geadresseerd. Deze adressering geschiedt vanuit de CPU.
De adresbus wordt altijd gebruikt in samenhang met de databus om de bron dan wel de bestemming aan te geven van de data die moet worden verzonden.

Controlbus
Via de controlbus worden besturingssignalen overgedragen van en naar de microprocessor en andere componenten (read, write, enz.). De adresbus en zeker de controlbus zijn weinig belangwekkend voor de computergebruiker (programmeur). Zij zijn wel van belang voor de computerbouwer. (Afbeelding 13.2 geeft een overzicht van de totale computer.)



Afb. 13.2. Computerarchitectuur

De CPU zoals de programmeur hem ziet
Lang niet alle onderdelen van de CPU zijn interessant voor wie 'alleen maar' programma's wil schrijven die de CPU kan uitvoeren. Het is als met het dashboard van een auto; men kan het bedienen zonder precies te weten wat zich achter het dashboard afspeelt. Afbeelding 13.3 geeft een overzicht van wat men het 'dashboard van de CPU' zou kunnen noemen.



Afb. 13.3. De computer gezien vanuit de software

De programmeur moet dit kennen omdat hij de onderdelen die in dit schemd zijn genoemd, rechtstreeks kan aanwijzen in de programma-instructies. Dat geldt (natuurlijk) voor de accumulator (ook wel genoemd register A), het geldt ook voor de registers B tot en met L, waarin hij waar nodig operandgegevens kan opslaan (die dan erg snel en erg gemakkelijk bereikbaar zijn). Daarnaast moeten vanuit het programma de conditieflags bereikbaar zijn. Waar nodig moet de inhoud van de programmateller vanuit het programma bepaald kunnen worden en moet de stackpointer toegankelijk zijn. Achter dit beperkte aantal onderdelen gaat de gecompliceerde CPU verder schuil.
Als de programmeur zich van de genoemde onderdelen op correcte wijze bedient gaat de rest metterdaad automatisch.

13.3. Geheugen

Het systeem heeft diverse functies. In de eerste plaats moet er in het computersysteem een schakeling zijn, die de te bewerken gegevens kan bevatten en waarin tussenresultaten en eindresultaten tijdelijk kunnen worden opgeborgen. Ook programma-instructies moeten ergens in de computer staan en wel in het geheugen. Een geheugen moet de volgende eigenschappen hebben.

- elk gewenst bitpatroon moet er in kunnen worden geschreven;
- deze patronen moeten er blijven staan tot ze opzettelijk worden veranderd (overgeschreven);
- de bitpatronen moeten, indien nodig, kunnen worden uitgelezen.

Programma-instructies en gegevens zijn meestal georganiseerd in patronen van een vaste lengte (bij voorbeeld een byte). Het geheugen is daarom in dezelfde woordlengte georganiseerd. Elk woord in het geheugen heeft een nummer (het adres). De CPU stuurt een adres en een stuurcommando (READ of WRITE) naar het geheugen en hierdoor wordt in het geheugen op de aangegeven plaats een woord gelezen of geschreven. (Dit aanwijzen gebeurt door het adresselectiemechanisme.) Behalve de normale lees/schrijfgeheugens zijn er ook speciale geheugens die alleen kunnen worden gelezen. Read Only Memory is hiervoor de vrij algemene benaming of kortweg ROM. Buiten de computer zijn ze wel te beschrijven, zij het met speciale apparatuur. Ze worden gebruikt voor opslag van vaste programma's of vaste gegevens. De microcomputer die het onderwerp van dit boek vormt kent ROM-geheugen naast geheugen waarin zowel gelezen als geschreven kan worden. Het laatste krijgt de benaming RAM (Random Acces Memory). De omvang van het geheugen staat niet vast. Wel de maximale grootte: 65 536 woorden, een aantal dat bepaald wordt door het feit dat het adresregister zestien bits telt. RAM en (P)ROM vormen samen 'het' geheugen.

13.4. Input en output

De CPU kan data met de buitenwereld uitwisselen door middel van I/O-modules. De functies van deze modules zijn:

- in geval van output; een byte die wordt uitgezonden door de CPU, opslaan in een outputregister (outputpoort);
- bij input; het opnemen van signalen die op de inputlijnen van een outputpoort (buffer) staan en deze doorgeven naar de CPU.

Evenals bij het geheugen is er bij de I/O een selectiemechanisme om de juiste poort vóór deze data-uitwisseling te adresseren. Om output mogelijk te maken moeten de volgende handelingen door de I/O-logica worden gerealiseerd:

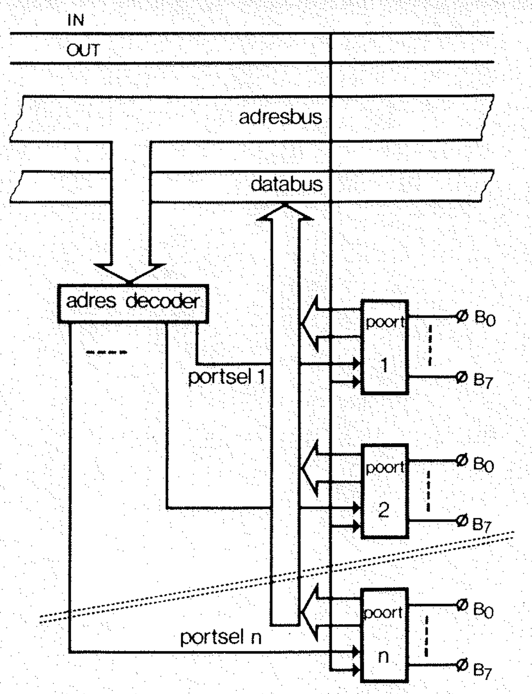
- de outputpoort(register), dat het acht bits woord van de CPU moet ontvangen, moet worden geselecteerd;
- via het OUT-commando moet worden aangegeven wanneer de data, uitgezonden door de CPU, op de databus staan;
- de outputpoort (register) moet de data overnemen.

Bij input zijn soortgelijke handelingen te onderscheiden, namelijk:
- de inputpoort moet worden geselecteerd;
- via een IN-commando moet het juiste ogenblik worden aangegeven wanneer de poort de data moet doorgeven naar de CPU.

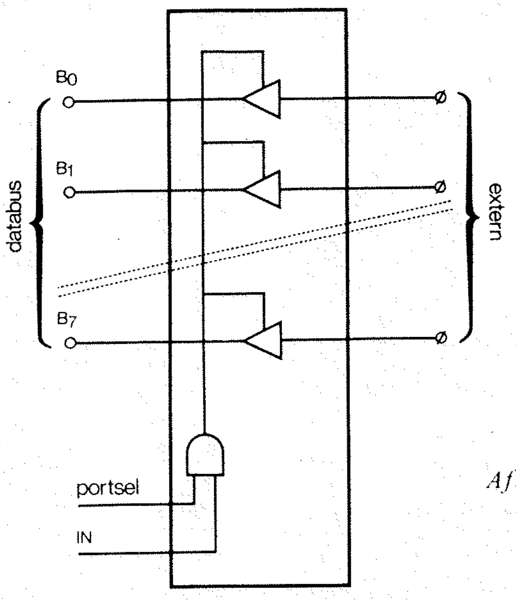
Er worden nogal wat eisen aan de flexibiliteit van de I/O-interface van een microcomputer gesteld. Het gaat niet alleen om het uitwisselen van signalen met meters, alarmapparaten en dergelijke. Er moeten ook randapparaten kunnen worden aangesloten zoals ponsbandapparaten, een toetsenbord, een beeldscherm, floppy-disk, enz. Een microcomputer zoals het 8080-systeem moet dat allemaal aan kunnen. Qua structuur kan het I/O-systeem het best als een rij éénbyte registers worden beschouwd, waar data in geschreven en uit gelezen kunnen worden. De 8080 processor kent hiervoor speciale instructies: IN en OUT. Over het algemeen gaat het om eenvoudige transporten tussen de accumulator en een geadresseerde input/outputpoort. Bij input worden de data die via de gespecificeerde inputpoort op de databus zijn geplaatst, naar de accumulator getransporteerd. Bij output worden op grond van de OUT-instructie de data uit de accumulator naar de geadresseerde outputpoort overgebracht.

Input-buffers en Output-latches

De input/outputpoorten dienen voor het uitwisselen van data tussen de microcomputer en externe apparatuur. Een inputpoort laat data (bij selectie) door naar de databus. De selectie vindt plaats als het adres op de adreslijn overeenkomt met het poortnummer en er een besturingssignaal wordt gegeven (nl. het signaal IN). Deze beide condities geven aan dat op dat moment de externe data moeten worden doorgegeven (zie afbeelding 13.4).
De inputpoort wordt gevormd door acht schakelingen die externe signalen al of niet doorlaten naar de interne databus. Zo'n schakeling

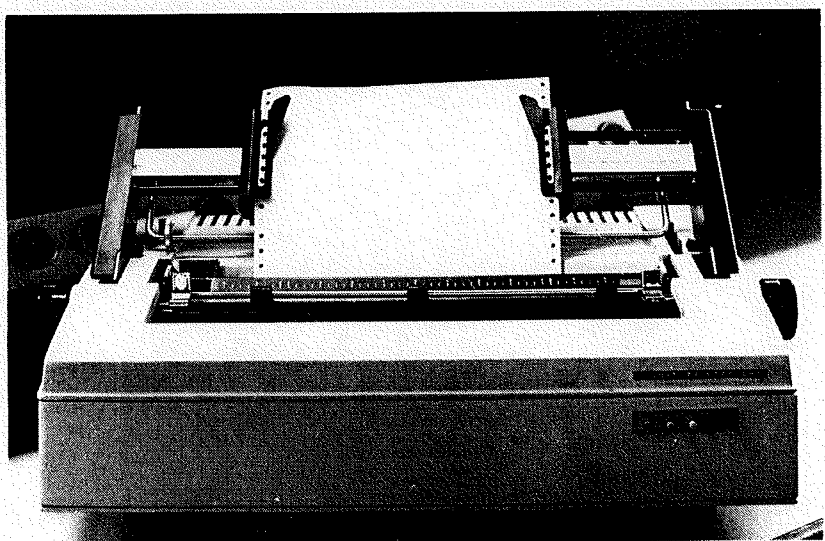


Afb. 13.4. Inputpoorten in een computer

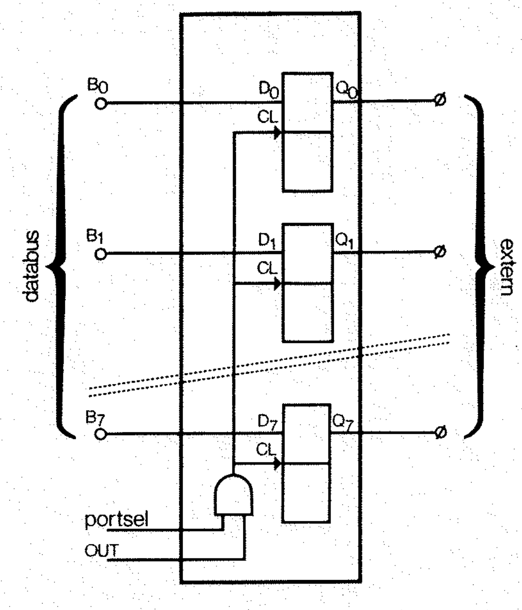


Afb. 13.5. 8 bits inputpoort

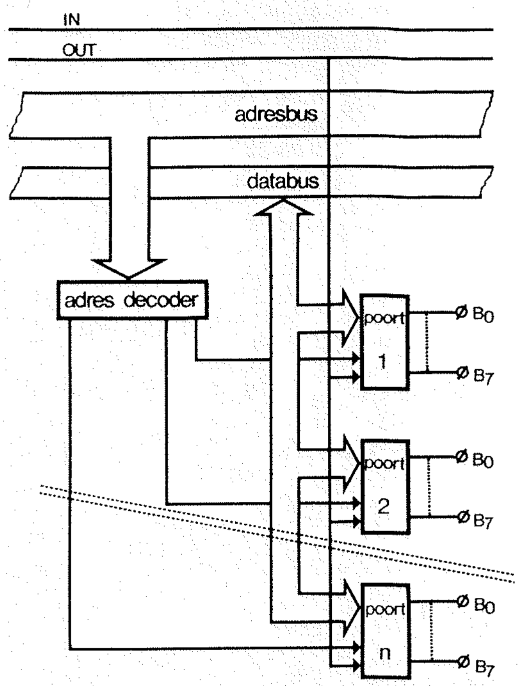
is dus te vergelijken met een sluis. Afbeelding 13.5 zet de werking ervan in schema; voor de betreffende schakeling wordt in de elektronica het driehoeksymbool gebruikt.
Een outputpoort functioneert in grote lijnen op dezelfde wijze als een inputpoort, zij het dat de data van de accumulator naar buiten moeten worden gebracht. Een probleem daarbij is dat de data door de buitenwereld moeten worden overgenomen en het is niet te voorspellen wanneer dat zal gebeuren. De kans is groot dat dat niet precies op het moment zal zijn dat de data beschikbaar komen. We moeten daarom de outputpoorten zo opbouwen dat ze de toegewezen data vasthouden. Daarvoor worden zogenaamde *latch-flipflop-schakelingen* gebruikt. Deze accepteren de data op het moment dat ze worden geselecteerd en houden ze vast. Tussen de beide selectiemomenten blijven de data beschikbaar voor de buitenwereld. Af-



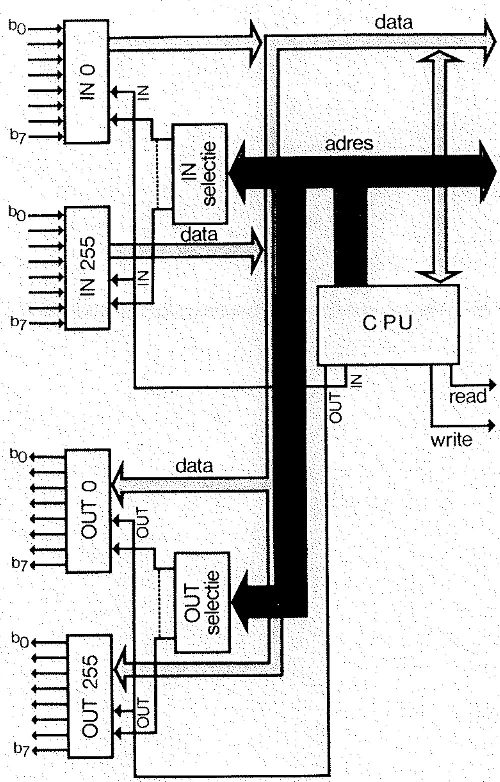
Voorbeeld van een printer



Afb. 13.6. 8 bits outputpoort



Afb. 13.7 Outputpoorten in een computer

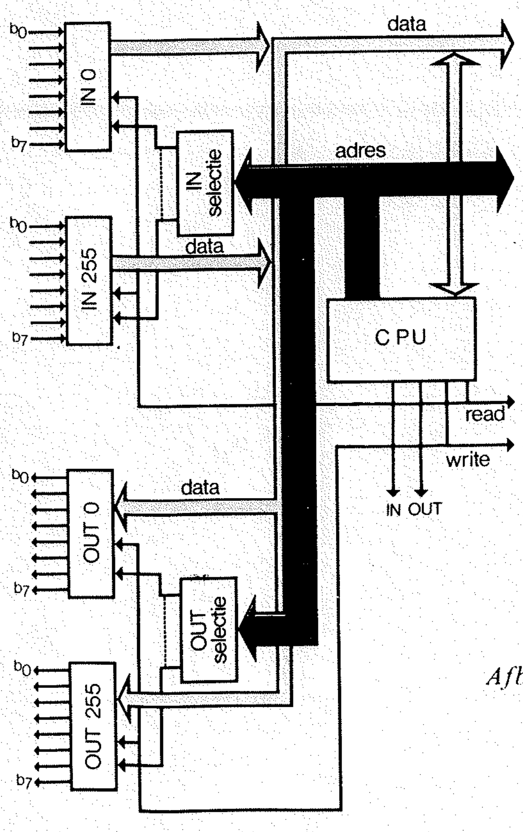


Afb. 13.8. Een I/O-mapped I/O systeem

beelding 13.6 geeft van een output-latch de symbolische weergave; afbeelding 13.7 geeft een schematisch overzicht van het output-proces.
Het complete input-/outputsysteem is weergegeven in afbeelding 13.8.

Adressering van I/O-poorten
Data-uitwisseling vanuit de microprocessor naar buiten en binnen (een I/O-poort) geschiedt op woordbasis. Een woord zal hierbij meestal de breedte hebben van de databus, maar dit is niet noodzakelijk. Voor het gemak zullen we toch hiervan uitgaan. Om data met de buitenwereld te kunnen uitwisselen, zijn er wat betreft de adressering twee systemen in gebruik, namelijk *I/O-mapped-I/O* en *memory mapped I/O.*
Het verschil in de twee systemen ligt in de manier van plaatsing van de I/O-poorten op de besturingsbus.
Bij I/O-mapped-I/O is de selectie van een I/O-poort in de hardware afgeleid van een adres en van een besturingssignaal IN en OUT (I/O-read) (I/O-write). Deze worden door de processor gegenereerd bij de uitvoering van de INPUT en OUTPUT instructies. Het is hierbij mogelijk om bijvoorbeeld via een OUTPUT instructie met daarbij een poortnummer een byte vanuit de accumulator naar de outputpoort te sturen.
In geval van memory-mapped-I/O wordt de selectie van de poorten uit een adres en de besturingssignalen READ (memory read) en WRITE (memory write) afgeleid.
Nu moet het datatransport in een programma door een memory-reference instructie worden gestuurd zoals bijvoorbeeld MOV M, A of MOV A, M voor respectievelijk output of input.

Het verschil voor de gebruiker in de diverse modes is dat er andere instructies worden gebruikt voor de I/O-datatransport. Er zijn in de I/O-mapped mode 256 input- en 256 outputpoorten. In de memory-



Afb. 13.9. Een memory-mapped I/O systeem

mapped mode worden de I/O-poorten als het ware op geheugenplaatsen gezet. We kunnen in principe alle 16 bits als adres gebruiken voor I/O-poorten. (Er moet echter wel een deel van de normale geheugenfuncties beschikbaar blijven om te kunnen werken.) De input- en outputoperaties werken met behulp van normale geheugen-(lees en schrijf-)instructies.
Een ander verschil is het feit dat de IN en OUT instructies alleen samenwerken met de accumulator. Bij de memory-mapped-I/O kunnen we meerdere registers als bron (of bestemming) hebben. We kunnen zo dus meerdere bronnen voor I/O-transfer door elkaar gebruiken. In een microcomputersysteem is gewoonlijk voor een van beide methodes gekozen. Als programmeur moet u weten welke methode is gebruikt opdat u de juiste instructies kunt gebruiken.

Voorbeeld
We willen data sturen naar de outputpoort die staat op adres 0F000H (F000 hexadecimaal).
Om het transport te kunnen realiseren moeten we zorgen dat het juiste adres gegenereerd wordt in de bijbehorende I/O-opdracht. Stel dat de data in register C staat. Het programma kan nu zijn:

```
LXI H, 0F000H
MOV M, C
```

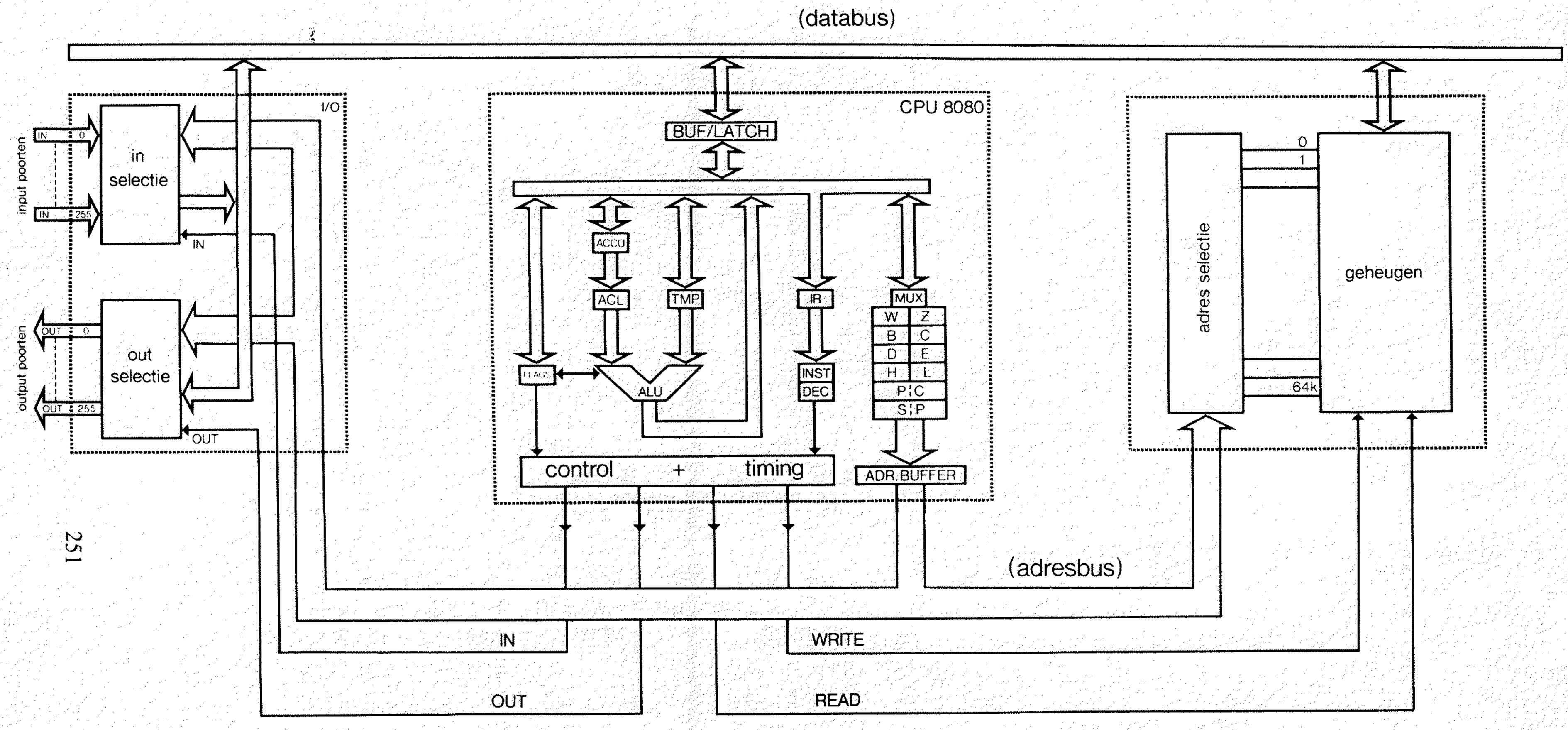
of een ander programma met hetzelfde effect:

```
MOV A,C
STA 0F000H
```

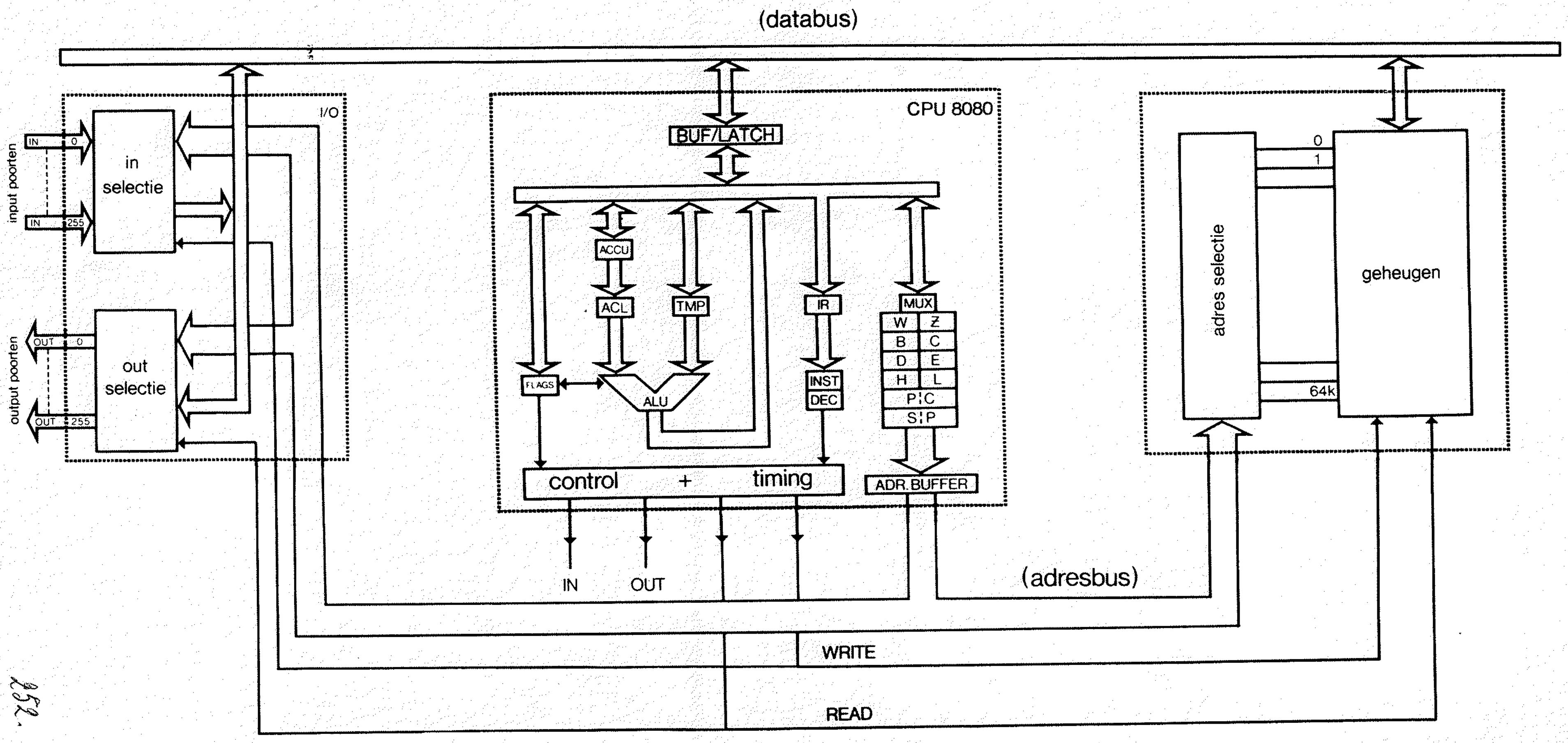
In het eerste voorbeeld maken we gebruik van de geheugenpointer (H,L). Deze wordt door de load registerpair immediate instructie (LXI) gevuld met het adres van de betreffende I/O-poort. Daarna wordt door middel van een move register to memory het transport gerealiseerd. In dit geval is het registerpaar H,L dus veranderd. Zie hoofdstuk 12 voor een overzicht van de instructies. Bij het tweede voorbeeld worden H en L niet veranderd echter de inhoud van de accumulator wel, omdat we een move accumulator direct instructie gebruiken voor het datatransport. Dit kan alleen vanuit de accumulator. Deze moet dus eerst als inhoud de te transporteren data krijgen. Wat betreft de programmagrootte is er geen verschil. In het eerste geval worden echter de registers H en L veranderd terwijl in het tweede geval alleen register A verandert.
Ga zelf eens na of er snelheidsverschillen tussen beide methodes bestaan. Gebruik het overzicht van hoofdstuk 12 daarbij.
Voor een input-routine, die vanuit poort 0F100H data haalt en deze in register B zet, kunnen we het onderstaande programma gebruiken:

```
LDA 0F100H
MOV B,A
```

Nu is zowel de waarde van register A als B veranderd.



Afb. 13.10a. Een computersysteem met I/O-mapped I/O



Afb. 13.10b. Een computersysteem met memory-mapped I/O

Samenvatting hoofdstuk 13

In dit hoofdstuk is de structuur van een microcomputer besproken. Verder is een tweetal methodieken besproken om I/O in een microcomputersysteem te realiseren.

Multiple choice-vragen bij hoofdstuk 13

Vragen over de hoofdstukken 1 t/m 12

A. **a.** We willen bepalen of een getal een geheel veelvoud is van 3. We doen dit op correcte wijze m.b.v. de volgende relationele expressie:

3 * INT (A/3) = A

b. In het statement:
P = ASC(CHR$(Q))
wordt P identiek aan Q voor toegestane waarden van Q.

1 uitspraak a is goed en uitspraak b is goed
2 uitspraak a is fout en uitspraak b is goed
3 uitspraak a is goed en uitspraak b is fout
4 uitspraak a is fout en uitspraak b is fout

B. **a.** De programmaregel:
10 Q = ABS(P)
is identiek aan:
10 IF SGN(P) = −1 THEN Q = −P

b. We willen een getal (A) afronden.
Als de fractie kleiner dan 5 is, ronden we af naar beneden.
We doen dit correct op de volgende wijze

INT(A) + 0.5

1 uitspraak a is goed en uitspraak b is goed
2 uitspraak a is fout en uitspraak b is goed
3 uitspraak a is goed en uitspraak b is fout
4 uitspraak a is fout en uitspraak b is fout

Onderstaand programma hoort bij de opgaven 13 C, D, E en F

```
 5 BIJ DIT PROGRAMMA MOET ER EEN STRING WORDEN
   INGEVOERD, WELKE O.A. DE LETTERCOMBINATIE AA
   BEVAT.
10 INPUT A$: PRINT
20 FOR I = Ø TO LEN (A$)−2
30 IF MID$ (A$, 1, 2) < > 'AA' THEN 50
40 PRINT A$, I: GOTO 60
50 NEXT I
60 END
```

C. Als de waarde van I wordt afgedrukt, is de eerste letter van de string AA gevonden op stringpositie:

1 I−1
2 I
3 I+1
4 2

D. Het programma zal ook andere combinaties AA aanwijzen als we in regel 40 zetten:

1 40 PRINT A$, I : GOTO 20
2 40 PRINT A$, I : GOTO 30
3 40 PRINT A$, I : GOTO 50
4 40 PRINT A$, I : GOTO 60

E. a. Als we in regel 30 de string AA vervangen door twee cijfers, dan zal niet de plaats van deze cijfers worden bepaald.
b. Als we in regel 30 de string AA vervangen door de string AAA kunnen we hierdoor een foutmelding krijgen.

1 uitspraak a is goed en uitspraak b is goed
2 uitspraak a is fout en uitspraak b is goed
3 uitspraak a is goed en uitspraak b is fout
4 uitspraak a is fout en uitspraak b is fout

F. We vervangen regel 30 door:
30 IF RIGHT$(LEFT$(A$,I),2) < > "AA" THEN 50

1 we krijgen een foutmelding
2 slechts de eerste combinatie AA wordt gevonden
3 slechts de laatste combinatie AA wordt gevonden
4 dit werkt alleen voor cijfers en niet voor letterstrings

Onderstaand programma behoort bij de vraagstukken 13G, H en I

```
 10 INPUT "OPLOSSING IS: ",B: PRINT
 20 READ K3
 30 DIM A(K3)
 40 FOR I = 0 TO K3-1
 50 READ A(I)
 60 NEXT I
 70 S = 0
 80 FOR I = 0 TO K3-1
 90 IF A(I) = B THEN S = S + 1
100 NEXT I
110 PRINT S/K3
120 DATA 13,1,2,3,4,4,4,3,1,4,1,2,4,4
130 END
```

Dit programma moet de gemiddelde score behalen bij een multiple choice-opgave. In regel 120 staat als eerste een getal, dat aangeeft hoeveel oplossingen er zijn ingestuurd en daarop volgen de ingestuurde oplossingen.

G. In het programma wordt in de regels 20 en 30 bepaald hoe de dimensie van het array A minimaal moet zijn.

1 dit gebeurt direct
2 er had in regel 30 moeten staan DIM A(K3−1)
3 er had in regel 30 moeten staan DIM A(K3+1)
4 dit kan nooit via dit principe

H. a. In regel 40 en 80 had in plaats van K3 – 1 als grenswaarde K3 moeten staan

b. In regel 110 had moeten staan PRINT S

1 uitspraak a is goed en uitspraak b is goed
2 uitsoraak a is fout en uitspraak b is goed
3 uitspraak a is goed en uitspraak b is fout
4 uitspraak a is fout en uitspraak b is fout

I. We vervangen de regels 40 t/m 100 door:

```
40 S=0
50 FOR J=1 TO K3
60 READ C
70 IF C=B THEN S=S+1
80 NEXT J
```

a. Dit programma heeft hetzelfde resultaat.
b. Ook de regels 20 en 30 mogen vervallen.

1 uitspraak a is goed en uitspraak b is goed
2 uitspraak a is fout en uitspraak b is goed
3 uitspraak a is goed en uitspraak b is fout
4 uitspraak a is fout en uitspraak b is fout

Onderstaand programma hoort bij opgave J.

```
 10 REM LOTING 6 UIT 49
 20 DIM A(6)
 30 FOR I=1 TO 6
 40 K = INT (49*RND (1))+1
 50 FOR J=0 TO I-1
 60 IF A(J)=K THEN GOTO 40
 70 A(I)=K
 80 NEXT J
 90 NEXT I
100 END
```

J. In regel 40 genereren we getallen van:

1 0 tot en met 49
2 1 tot en met 49
3 0 tot en met 50
4 1 tot en met 50

Hoofdstuk 14

Interface-technieken

In dit hoofdstuk wordt besproken hoe met de input- en outputinstructies van een microcomputer een koppeling met een randapparaat kan worden gerealiseerd. Deze koppeling wordt wel interface genoemd. Hiervoor wordt een analyse gemaakt van de acties, die moeten worden uitgevoerd door zowel de computer als het randapparaat om een foutvrije data-uitwisseling te realiseren. Tenslotte wordt ingegaan op het principe van seriecommunicatie tussen computer en randapparatuur.

14.1. Binaire codes

Er zijn vele soorten technieken voor de overdracht van signalen tussen twee systemen in gebruik, zoals bijvoorbeeld een computer en een randapparaat *(periferieapparaat)*. Een algemeen gebruikte manier is het toepassen van digitale signalen. Deze zogenaamde binaire (tweewaardige waarden) signalen zijn via elektronische hulpmiddelen eenvoudig te realiseren (bijvoorbeeld d.m.v. schakelaars, relais, transistoren enz.). Door een aantal van deze binaire signalen te combineren is het mogelijk codes te vormen, die symbolen voorstellen. Deze symbolen moeten zo worden gekozen dat ze voor zowel de *producent* als de *consument* een gelijke betekenis hebben. Voor elk mogelijk symbool kunnen we een andere binaire code kiezen. Als we slechts een binaire *digit* (cijfer) gebruiken zijn er 2 (= 2 ↑ 1) combinaties mogelijk. Namelijk één voor de waarde 0 en één voor de waarde 1. In plaats van één digit kunnen we er ook meer nemen. Bij n bits hebben we 2 ↑ n combinaties of codewoorden. Voor n = 3 hebben we 8 mogelijkheden, namelijk:

000
001
010
011
100
101
110
111

Voor de symbolen zullen in het algemeen de volgende mogelijkheden worden gebruikt:

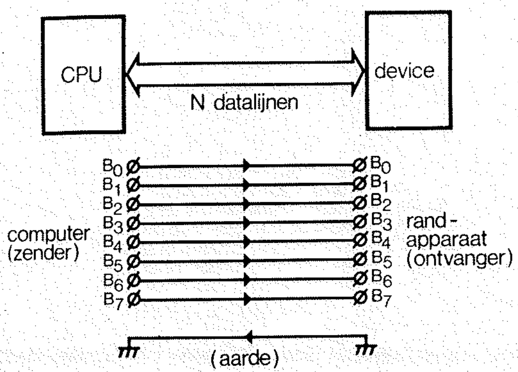
- letters;
- cijfers;
- leestekens;
- besturingstekens, zoals nieuwe regel, wagenterugloop, spatie e.d.

In de praktijk worden bij de data-uitwisseling tussen apparaten meestal codewoorden van 8 bits gebruikt. De bits in deze woorden worden als volgt genummerd:

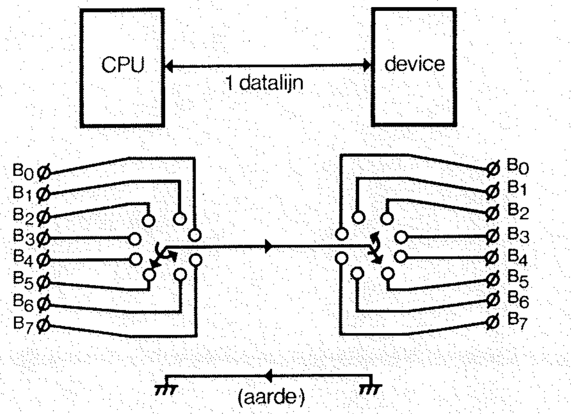
b7 b6 b5 b4 b3 b2 b1 b0

Om de symbolen tussen apparatuur te transporteren zijn er twee algemene technieken in gebruik, namelijk:
- parallel data-uitwisseling;
- serie data-uitwisseling.

Bij de parallelle methode is er voor elk bit van het codewoord een draad *(kanaal)*. Het codewoord zal daarover in een keer worden getransporteerd (parallel). (Zie afbeelding 14.1 voor een parallel kanaal van 8 bits.) Bij de serie data-uitwisseling wordt het codewoord over een enkele draad verzonden en wel bit voor bit, dus in serie. Afbeelding 14.2 laat een seriekanaal zien. Let op dat er bij beide tekeningen een aarddraad aanwezig is als elektrische niveaureferentie, opdat zowel de zender en ontvanger van de gegevens spanningsverschillen kunnen vaststellen.



Afb. 14.1. Een parallel datakanaal



Afb. 14.2. Een serie datakanaal

14.2. Interface

In de voorgaande hoofdstukken is globaal de werking van de centrale processor, het geheugen en de I/O poorten behandeld. Om deze 'bouwstenen' te kunnen gebruiken voor besturing van randapparaten, moeten we ons eerst bezinnen op de werking van deze apparatuur. Er is een besturingsprotocol nodig voor een juiste datauitwisseling. Het aansluitschema en de wijze van besturen wordt *interface* genoemd. Een interface bestaat namelijk uit verschillende onderdelen. We kunnen het protocol als het ware een voorschrift noemen voor het uitwisselen van data. Maar ook voorschriften voor wat betreft de elektrische signaalniveaus, volgens welke de randapparaten moeten worden gestuurd. Een ander deel van een interface zijn de mechanische constructieregels, waaraan moet zijn voldaan bij het koppelen van apparatuur. Hierop zullen we echter niet ingaan. Het belangrijkste punt dat we in dit hoofdstuk bespreken is de realisatie van het interface-protocol volgens welke de data-

uitwisseling plaatsvindt. Een protocol is een beschreven afspraak over gegevens-uitwisseling. Hierbij zijn twee manieren toepasbaar.

De eerste manier is het besturen van de randapparatuur door software. Dit betekent het hoog en laag maken van de bits van I/O-poorten en het testen van de ingangssignalen met behulp van input- en output-instructies (of in geval van memory-mapped-I/O de overeenkomstige memory-instructies). Een andere mogelijkheid is het realiseren van een interface met behulp van speciale hardware schakelingen. Voordat we verder kunnen gaan moeten we ons eerst bezinnen op wat data-uitwisseling tussen computer en periferie-apparaat precies betekent.

14.3. Parallelle I/O

Een groot deel van de I/O-apparaten zullen een aantal parallelle signaallijnen hebben, die als volgt zijn te groeperen:
— datalijnen;
— besturings- (control- of commando-) lijnen;
— statuslijnen.

Datalijnen zijn die signaallijnen waarover de informatie, dus de codewoorden tussen de computer en het randapparaat, worden uitgewisseld. We kunnen hierbij een onderscheid maken tussen apparatuur, die ontvangt van de computer *(outputapparatuur)* of verstuurt naar de computer *(inputapparatuur)*. Maar er zijn ook apparaten, die zowel ontvangen als zenden. Bij de naamgeving gaan we altijd uit van de CPU. Dit wil zeggen dat ontvangen betekent dat de CPU ontvangt (dus input). Output betekent vanuit de CPU naar buiten. Een apparaat dat zowel kan ontvangen als zenden heet *bidirectioneel.* Om te zorgen dat zowel de computer als het randapparaat van elkaar weten dat de data-uitwisseling goed gebeurt, zijn er naast de datalijnen nog een aantal besturings- en terugmeldingslijnen.

Besturingslijnen
Om een randapparaat een of andere functie uit te laten voeren, moeten we deze functie aanvragen (of opdrukken) door middel van een commando. Enkele mogelijke commando's zijn:

Geef data
De computer vraagt aan het randapparaat om inputdata beschikbaar te stellen.

Reset apparaat
Als er in een randapparaat een fout is opgetreden, zal dit door het randapparaat worden gemeld aan de computer. De computer zal daarna de nodige acties ondernemen om de foutconditie te herstellen (of door de operator te laten herstellen). Zodra de oorzaak van de fout is weggenomen, kan de computer als opdracht aan het device (randapparaat) geven de foutmelding weg te nemen (te 'reset'ten). Meestal wordt deze opdracht ook gegeven bij het starten van het computersysteem om zeker te zijn van de situatie waarin een randapparaat verkeert.

Verander draairichting
Wanneer van een ponsbandlezer gebruik wordt gemaakt is het soms nodig om een ingelegde ponsband terug te spoelen naar het begin van de band. De band kan daarna weer opnieuw worden gelezen door de computer. Hiervoor moet de computer tweemaal de draairichting veranderen. Eerst achterwaarts *(backward)* en daarna voorwaarts *(forward)*.

Spoel cassette terug
Dit signaal is te vergelijken met het voorgaande voorbeeld, maar nu geldt het voor een cassetteband. Dit commando zal altijd worden gegeven voordat een cassette uit de machine kan worden genomen. Zo kan de gebruiker namelijk nooit per ongeluk met zijn vingers aan de magnetische band komen.

Ga naar het volgende record
Bij een tape-unit zal de machine telkens blokken informatie schrijven. We schrijven dan niet 1 byte, maar groeperen de bytes tot blokken. Zodra we deze informatie willen gebruiken, kan het gebeuren dat we van het ene naar het andere blok moeten gaan. We kunnen daarvoor het zojuist genoemde commando geven.

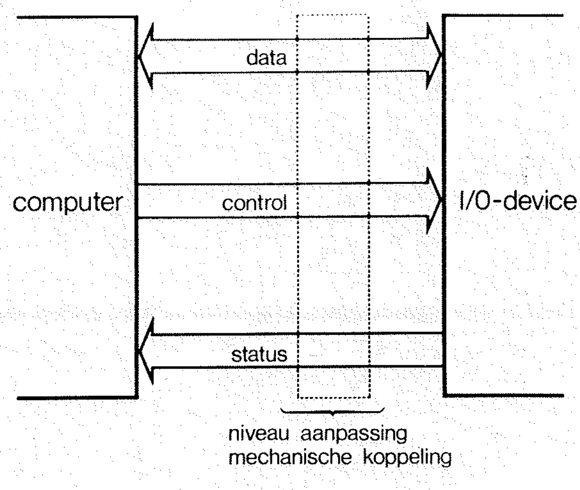
Seek track
Bij een floppy disk moeten we soms de leeskop van de ene naar de andere track brengen. Een track is als het ware een cilinder op de schijf. Hiervoor kunnen we aan het device de opdracht geven om de leeskop naar een gegeven track (spoor) te brengen. Bijvoorbeeld Seek track 20.

Load head
Om tijdens de Seek opdracht de leeskop niet onnodig warm te laten worden door wrijving tegen de floppy disk, zal de kop automatisch van de disk worden gemaakt. Zodra we op de gewenste plaats (track) zijn aangekomen, moeten we de kop weer 'laden' (op de disk brengen).
Het is duidelijk dat er bij andere typen randapparaten behalve deze signalen nog tientallen andere signalen mogelijk zijn. Het zou in dit bestek te ver voeren om al deze varianten te bespreken. Het is nodig om dit te doen als men met een bepaald type apparaat te maken krijgt dat moet worden aangesloten. Het is wel nuttig om eens de gedachten te laten gaan over het hele interface-spel dat gespeeld moet worden, zodat we een duidelijk begrip krijgen van wat er zoal nodig is om een periferie-apparaat te besturen. Zie ook hoofdstuk 17.

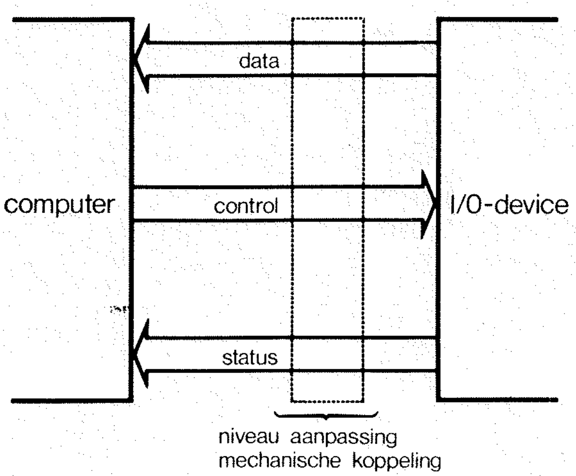
Terugmeldingssignalen
Naast de commando- of controlsignalen zal de interface tussen een computer en een randapparaat nog status(terugmeldings)-signalen bevatten. De statuslijnen lopen vanuit het device naar de computer. Dit in tegenstelling tot de commandosignalen, die vanuit de computer komen. De functie van de statussignalen is om aan de computer te melden in wat voor een toestand *(state)* het randapparaat zich bevindt. De overgang van de ene naar de andere state kan het gevolg zijn van uitgevoerde acties, mechanische storingen, de onmogelijkheid om een commando uit te voeren e.d.

Mogelijke statussignalen kunnen zijn:
— er is een fout opgetreden;
— de tape staat op de aanloopstrook, d.w.z. de data zijn niet goed overgekomen;
— er is een transmissiefout gevonden;
— de data is overgenomen;
— de inputdata is geldig;
— er is een mechanische storing.

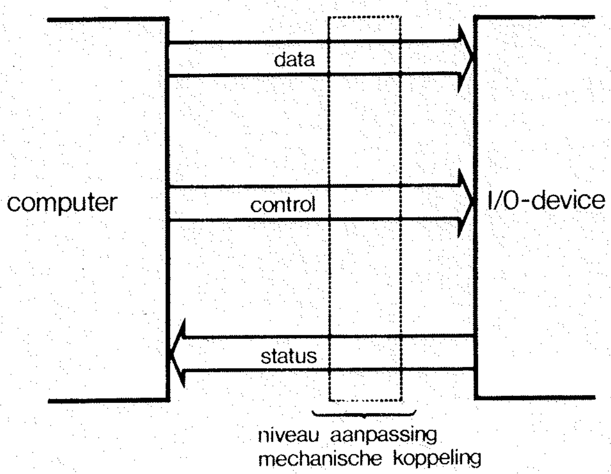
Interface-protocol
Het deel van de interface, dat we nu bespreken, gaat over de afspraken, waarin vastgelegd wordt hoe de signaaluitwisseling moet zijn om een datatransport tussen randapparaat en computer te verwezenlijken. Deze afspraken noemen we het interface-protocol. Het zou zinvol zijn om een interface-protocol van een of ander randapparaat door te lezen en uit te pluizen. Het zal blijken dat de interface tussen een computer en een randapparaat echt niet zo moeilijk is als men vaak moet voorkomen. Afbeelding 14.3 laat globaal zien hoe de interface-aansluitingen tussen een randapparaat en een computer lopen. De betekenis van de diverse signalen moeten voor een specifiek apparaat nader worden ingevuld. In de afbeeldingen 14.4 en 14.5 zien



Afb. 14.3. Algemeen interfaceschema

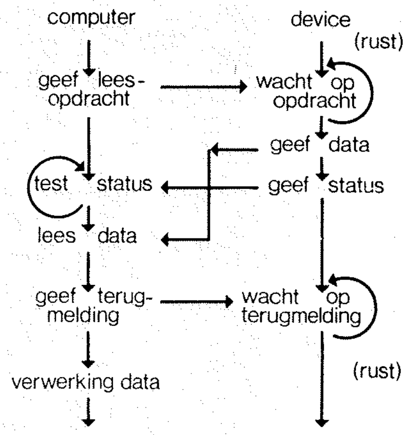


Afb. 14.4. Een input interface



Afb. 14.5 Een output interface

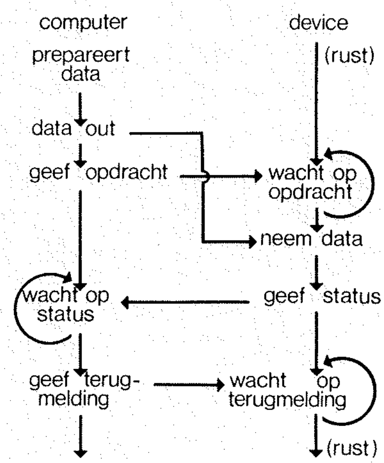
we achtereenvolgens een inputdevice en een outputdevice, die zijn aangesloten op de I/O-poorten van een microcomputer.
Als we de interacties tussen de computer en het I/O-apparaat bekijken, kunnen we daarbij twee aspecten onderscheiden. Het ene aspect omvat het complex van handelingen, dat de computer zal moeten uitvoeren. Het andere geeft aan wat er zoal door het randapparaat gedaan moet worden. We zullen nu dit spel eens nader uitwerken. We gebruiken daarbij afbeelding 14.6. Deze afbeelding brengt de interacties schematisch in beeld. De linkerkant van de afbeelding laat zien wat de computer doet. Aan de rechterkant zien we de acties van het I/O-apparaat. Daartussen zijn de onderlinge signalen aangegeven.



Afb. 14.6 Een handshakeprotocol voor input

Stel dat we data van het inputapparaat willen lezen. We gaan er vanuit dat zowel de computer als het randapparaat zich in rust bevinden. Dat wil zeggen dat de computer een of ander programma zal uitvoeren en dat het randapparaat op een commando wacht. In de afbeelding wordt dit aangegeven door de tekst 'wacht op opdracht'. De rondgaande pijl toont dat het device als het ware telkens naar de computer kijkt of deze een bepaald commando uitzendt. Als de computer in het programma een routine tegenkomt waarin om een of andere reden data nodig is, zal een leesopdracht worden gegeven. We zien in de afbeelding dat er een leesopdracht in een randapparaat binnenkomt. Daarna wordt er door het device aan de computer (eventueel na een mechanische vertraging) data aangeboden. De computer weet niet wanneer deze data geldig (stabiel) is. Elektrisch zullen de datasignalen enige tijd moeten uitslingeren voordat ze stabiel zijn. Om aan te geven wanneer dit is gebeurd (meestal ligt deze tijd van tevoren vast), zal het randapparaat een statussignaal uitzenden. De data blijft hierbij steeds staan. We zien dat de computer telkens deze status bekijkt (test). Als de status aangeeft dat de data geldig is (data valid), zal de computer het datawoord inlezen. Nu mag het randapparaat de data wegnemen. Het randapparaat weet dit echter niet. Er moet namelijk eerst door de computer een terugmelding worden gegeven om te melden dat deze de data (in goede staat) heeft overgenomen. Dit interface-spel geeft duidelijk aan dat hier zowel het randapparaat als de computer exact van elkaar moeten weten wat elkaars status is voordat de data mag worden weggehaald. Het zal duidelijk zijn dat hierdoor het interface-spel foutvrij kan worden gemaakt. Dit hele interface-spel wordt in de computerwereld de *handshake* (handje schudden) tussen computer en randapparaat genoemd.

In afbeelding 14.7 zien we een soortgelijke handshake voor een outputapparaat. Hierbij verloopt de data-uitwisseling in tegengestelde richting. De computer stuurt de data naar buiten en geeft dan een commando wat betekent dat het randapparaat de data moet overnemen.
Het randapparaat zal een terugmelding geven zodra de data is overgenomen en daarin aangeven of er soms een fout bij de overdracht is opgetreden. De computer zal deze statussignalen testen, melden dat de data niet meer geldig is en de data wegnemen. Zowel randapparaat als computer bevinden zich daarna weer in een rusttoestand. Ga eens na hoe een en ander verloopt in afbeelding 14.7.



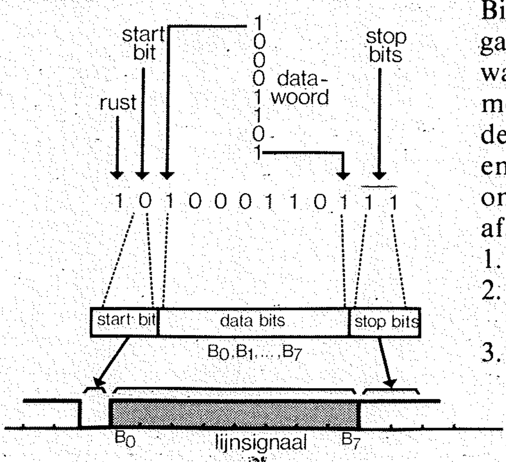
Afb. 14.7. Een handshakeprotocol voor output

14.4. Serie I/O

In de voorgaande paragrafen is alleen de parallelle interface aan de orde geweest. Veel randapparaten werken echter niet parallel maar in serie. Het is duidelijk dat er voor deze serie-overdracht van gegevens een speciale techniek moet worden toegepast. Bij de serietransmissie wordt telkens één element van het codewoord overgedragen in plaats van alle tegelijk zoals bij parallelle data-overdracht. Het wordt hierdoor mogelijk om codewoorden, die uit meerdere bits bestaan, over één enkel kanaal te versturen. Twee draden zijn hiervoor voldoende, namelijk een signaaldraad en een aarddraad. Een moeilijkheid hierbij is echter dat zowel de zender als de ontvanger steeds moeten weten met welke bit van een codewoord ze bezig zijn. Ze moeten *synchroon* lopen. Dit kan door een extra kloksignaal mee te geven of door extra afspraken.
In de zender en de ontvanger moeten codewoorden respectievelijk van parallel naar serie of van serie naar parallel worden omgezet (*geconverteerd*). Dit kan zowel met programma's als met extra hardware. In alle gevallen zullen deze extra's opwegen tegen de besparingen in kabelkosten. Een bijkomend aspect is de mogelijkheid om zo een standaard interface te maken. Enkele bekende standaarden zijn EIA RS 232 en V24 waarvan we in dit hoofdstuk de principes behandelen. Om de zender en de ontvanger te kunnen synchroniseren moet aan beide kanten de zogenaamde bittijd vastliggen. Dit is de duur van een bepaald bit op de interfacelijn. Deze bittijd wordt in praktijk de *baudrate* genoemd (op te geven in bits per seconde). Veel gebruikte snelheden zijn 110 baud voor teletypes, 1200 en 2400 baud voor datacommunicatie via normale telefoonlijnen en snelheden van 9600 baud voor rechtstreeks aan de computer gekoppelde videoterminals.

Zender
Bij de serie-interfaces komen enkele soorten protocollen voor. We gaan hier alleen in op de zogenaamde asynchrone serie-overdracht waarbij geen kloksignaal wordt meegezonden. Dit is namelijk de meest toegepaste methode voor wat betreft de randapparatuur. Bij deze serie dataoverdracht wordt het codewoord nog uitgebreid met enkele speciale bits. Er is namelijk nog een probleem: hoe weet de ontvanger wanneer een datawoord begint! Hiervoor is de volgende afspraak gemaakt.

1. Als er niets wordt verzonden is de lijn hoog.
2. Een nieuw woord wordt voorafgegaan door gedurende een bittijd een laag signaal uit te zenden (startbit).
3. Na een codewoord is de lijn gedurende 1 of 2 bittijden in rust (d.w.z. hoog). Dit zijn de stopbits.



Afb. 14.8. Parallel naar serie conversie (plus start/stopbits)

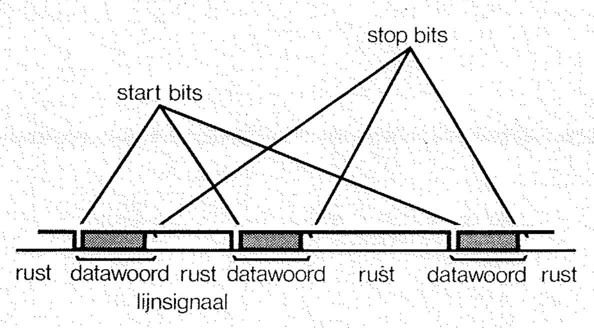
In afbeelding 14.8 zien we hoe het codewoord wordt omgezet in een signaal dat geschikt is voor serietransmissie.

Ontvanger
Bij het ontvangen van het character moeten we met de zojuist genoemde afspraken rekening houden. Wanneer we uitgaan van de rusttoestand, zal de ontvanger moeten testen op een overgang van

hoog (rust = stop) naar laag. Dit betekent dan dat er een startbit is ontvangen. Dit startbit moet gedurende een gehele bittijd laag zijn. Ook hierop moeten we testen, als we fouten helemaal willen uitsluiten. Na de detectie van een startbit zal de ontvanger telkens midden in een databit een monster (sample) van de lijn nemen. Zo wordt achtereenvolgens b0, b1 . . ., binnengehaald. We nemen telkens het monster midden in de bittijd om kleine snelheidsverschillen tussen zender en ontvanger te kunnen elimineren. Vaak is een codewoord 7 bits lang met daaraan toegevoegd een achtste (pariteits-) bit. Dit bit zorgt ervoor dat een woord altijd een even (of oneven) aantal enen bevat. Bij een even aantal enen spreken we van even pariteit. Bij een oneven aantal over oneven pariteit. Hiermee kan later nog een extra foutcontrole worden uitgevoerd. Aan het eind zal de lijn weer in de rusttoestand komen door de stopbits.

pariteitsbit	datawoord (7 bits) bij even pariteit
1	1001100
1	1111111
0	1111110
0	0000000

Om een codewoord van 8 bits (data plus pariteit) over te zenden moeten we dan bij voorbeeld 11 bits (woord plus start en 2 stopbits) oversturen. Bij de asynchrone transmissie is het gebruikelijk om van een datawoord vanaf het minst significante bit met oversturen te beginnen. In afbeelding 14.9 zien we het lijnsignaal bij het oversturen van een drietal woorden.



Afb. 14.9. Drie datawoorden op een serie datakanaal

Voordelen van asynchrone serietransmissie:
- eenvoudig te realiseren voor elektromechanische apparatuur;
- de karakters kunnen asynchroon worden verstuurd, omdat zowel de computer als het randapparaat een eigen synchronisatiemechanisme bezit, dus weinig kabelkosten.

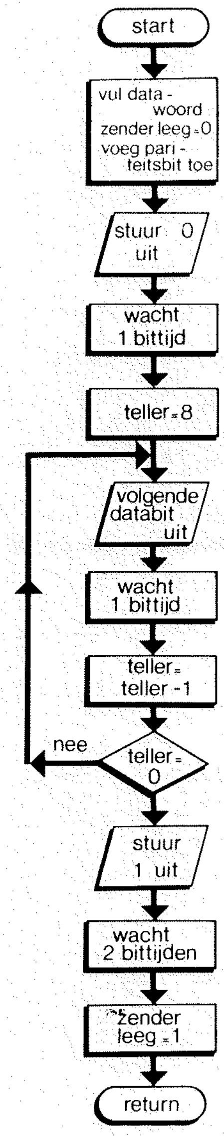
Nadelen van deze techniek zijn:
- de zender en de ontvanger hebben een eigen timing nodig;
- het systeem is gevoelig voor storing wat betreft het startbit;
- het mechanisme is niet efficiënt, omdat er minstens 2–4 extra bits nodig zijn om de code over te kunnen brengen. Er moeten namelijk altijd 1 startbit, een pariteitsbit en 1 of 2 stopbits worden toegevoegd.
- de overdracht is relatief langzaam om storingen in de timing te voorkomen.

Voor de code wordt meestal de ASCII-code gekozen. ASCII is een afkorting van: American Standard Code for Information Interchange.
Afbeelding 14.10 laat de ASCII-tabel zien voor woorden van 7 bits. De pariteits(parity-)bits moet afhankelijk van de keuze (even, oneven, altijd nul of altijd een) steeds afzonderlijk worden bepaald. We moeten wel oppassen voor wat de nummering van de bits betreft, want vaak wordt in plaats van met bit 0 vanaf b 1 genummerd. Bit 1 van de ASCII-tabel komt overeen met bit 0 van de computernummering.

ASCII-code: $b_7 b_6 b_5 b_4 b_3 b_2 b_1$

b_4 b_3 b_2 b_1	b_7 → / b_6 → / b_5 →	0 0 0	0 0 1	0 1 0	0 1 1	1 0 0	1 0 1	1 1 0	1 1 1
0 0 0 0		NUL	DLE	SPACE	0	@	P		p
0 0 0 1		SOH	DC1	!	1	A	Q	a	q
0 0 1 0		STX	DC2	"	2	B	R	b	r
0 0 1 1		ETX	DC3	#	3	C	S	c	s
0 1 0 0		EOT	DC4	$	4	D	T	d	t
0 1 0 1		ENQ	NAK	%	5	E	U	e	u
0 1 1 0		ACK	SYN	&	6	F	V	f	v
0 1 1 1		BEL	ETB	'	7	G	W	g	w
1 0 0 0		BS	CAN	(	8	H	X	h	x
1 0 0 1		HT	EM	)	9	I	Y	i	y
1 0 1 0		LF	SUB	*	:	J	Z	j	z
1 0 1 1		VT	ESC	+	;	K	[	k	{
1 1 0 0		FF	FS	,	<	L	\	l	|
1 1 0 1		CR	GS	-	=	M	]	m	}
1 1 1 0		SO	RS	.	>	N	Λ	n	~
1 1 1 1		SI	US	/	?	O	—	o	DEL

Afb. 14:10. De ASCII-tabel

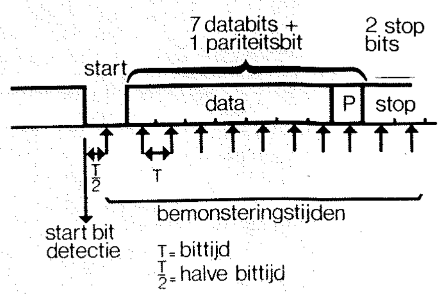


Afb. 14.11. Flow-diagram voor serie data-output

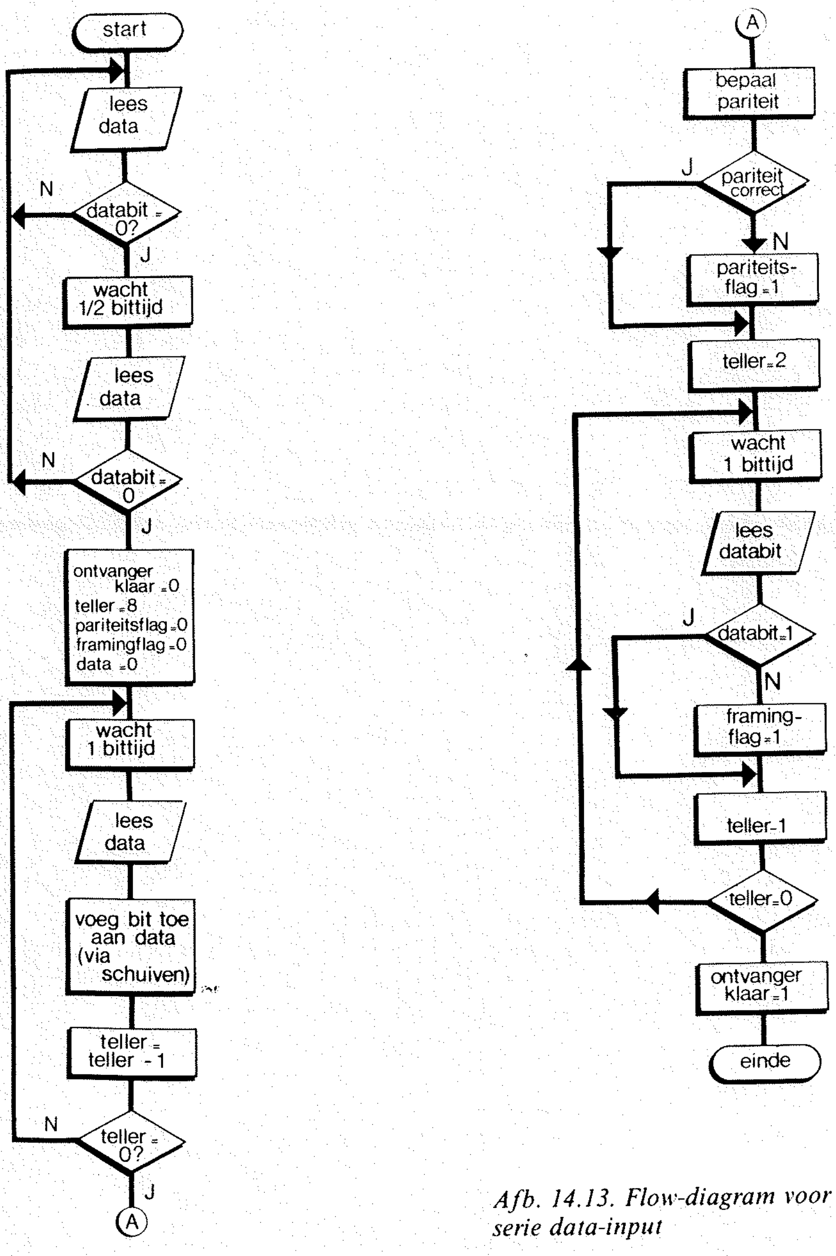
14.5. Asynchrone lijninterface

De bouw van een koppeling tussen een computer en een serie I/O-apparaat is nu erg eenvoudig geworden. De computer zal zowel als zender en als ontvanger dienst moeten doen. Als we een en ander met behulp van een programma willen realiseren, kunnen we als volgt te werk gaan. De computer zal bij het zenden eerst het startbit (laag) moeten versturen, daarna volgen van het datawoord de bits b0 tot en met b7 (waarbij b7 de pariteitsbit is), gevolgd door twee stopbits (hoog). Het algorithme dat dit moet realiseren zal via een geprogrammeerde wachtlus de bittijd voor de gewenste datarate realiseren. Het enige resterende probleem is het maken van een deelroutine die telkens het gewenste bit selecteert om te verzenden. Afbeelding 14.11 geeft hiervan een flow-diagram. In het begin van de routine wordt aan het datawoord het pariteitsbit toegevoegd. We zijn

dan klaar om de complete data naar buiten te sturen. Eerst het startbit (0) en daarna bit voor bit van het datawoord. Dit doen we door vanaf de minst significante plaats steeds een databit uit te zenden. We wachten hierbij steeds een bittijd, zodat de ontvanger steeds weet met welk bit we bezig zijn. Na het databit en het pariteitsbit volgt nog een hoog signaal gedurende twee bittijden (stopbits). Door middel van het signaal zender-leeg wordt aan het aanroepende proces doorgegeven dat het woord is uitgezonden.
Het ontvangen verloopt op een soortgelijke wijze. Het algorithme zal moeten testen of er een hoog naar laag-overgang op de lijn plaatsvindt. Dit betekent namelijk dat er een startbit is. We moeten daarna een halve bittijd later nog een keer testen of de startbit laag is (dat we dus niet te maken hebben met een kortstondig stoorsignaal). Is dit



Afb. 14.12. Bemonstering van serie inputdata

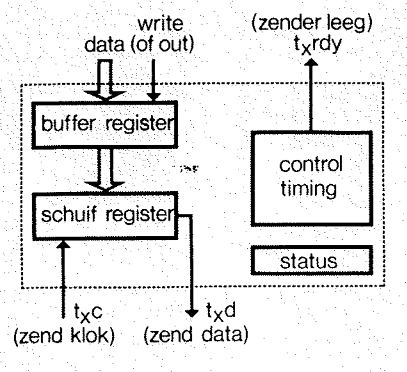


Afb. 14.13. Flow-diagram voor serie data-input

het geval dan kunnen we nu telkens een bittijd later de diverse bits van het woord en de stopbits binnenhalen; dit net zo vaak tot alle bits aanwezig zijn. Om te voorkomen dat er fouten ontstaan moeten we nog kijken of de pariteit van het ontvangen woord correct is en of beide stopbits zijn ontvangen. Vallen alle tests positief uit dan mag het ontvangen woord voor verder gebruik worden vrijgegeven. In afbeelding 14.12 zien we het ontvangen datawoord en de tijdstippen waarop we naar de lijn kijken, ofwel een monster nemen. In afbeelding 14.13 zien we een flow-diagram van het algorithme dat het ontvangen voor zijn rekening neemt. Nadat de routine is gestart zal steeds gekeken moeten worden of de datalijn gelijk aan 0 is. Is dit het geval dan moeten we een halve bittijd wachten. Daarna wordt nog eens getest of de datalijn 0 is. Dit is voor ons voldoende om aan te nemen dat een startbit is ontvangen. Nu zullen we achtereenvolgens 7 databits en een pariteitsbit binnenhalen. We schuiven telkens de nieuw ontvangen bit in een register. Dit heeft als resultaat dat zo bit voor bit aan het datawoord wordt toegevoegd. Dit doen we in een *programmalus*. In de lus wachten we telkens een hele bittijd. Hierdoor nemen we op de meest gunstige tijdstippen een monster van de datalijn. Als het pariteitsbit niet correct ontvangen is, wordt de pariteitsflag gelijk aan 1 gemaakt om aan te geven dat er een fout moet zijn opgetreden. Het binnenhalen van de stopbits gaat op dezelfde wijze. Als de stopbits fout zijn wordt de framing flag gelijk aan 1 gemaakt. Als laatste handeling zal nu de receiver ready (ontvanger klaar) bit 1 worden gemaakt. In het begin van de routine zijn de diverse flags op 0 gezet. We beginnen dan tenminste zonder foutmelding.

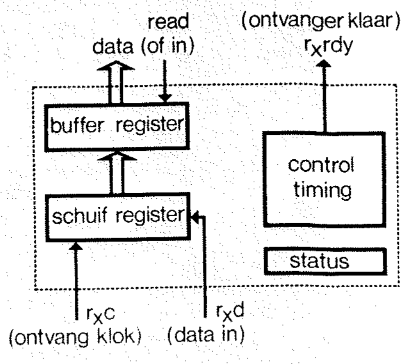
14.6. USART (Universal Synchronous/Asynchronous Receiver and Transmitter)

Uit het bovenstaande blijkt dat we de computer moeten laten testen op het binnenkomen van een startbit. Als dit bit niet wordt ontvangen, betekent dit dat de computer in de testlus blijft rondlopen. Hij is dus niet meer bruikbaar voor andere toepassingen. Om dit probleem uit de wereld te helpen, is door de halfgeleiderfabrikanten een chip ontworpen, die zich speciaal bezig houdt met ontvangen en verzenden van seriedata. Voor de programmatuur ziet deze chip er uit als een aantal registers, die onder programmabesturing toegankelijk zijn.
De registers hebben globaal de volgende functies. Twee registers doen dienst als respectievelijk zend- en ontvangstbuffer. Een derde register heeft de functie van statusregister. Door middel van een commandoregister is het mogelijk om vanuit de computer aan de USART een manier van werken op te geven. Hiertoe schrijft de computer in het commandoregister een commando. Dit commando geeft aan wat de baudrate moet zijn, hoe de pariteit is, hoeveel stopbits er moeten zijn, enz. Door de computer wordt in het zendbuffer een datawoord geschreven dat moet worden verzonden. De USART verzorgt dan zelf het toevoegen van start- en stopbits en het zenden met de juiste baudrate. Zodra de USART begint met het verzenden zal er in het statusregister een bit veranderen, dat aangeeft dat de zendbuffer leeg is. Wordt er een datawoord ontvangen, dan zal in het statusregister een ander bit veranderen, dat aangeeft dat er een woord staat

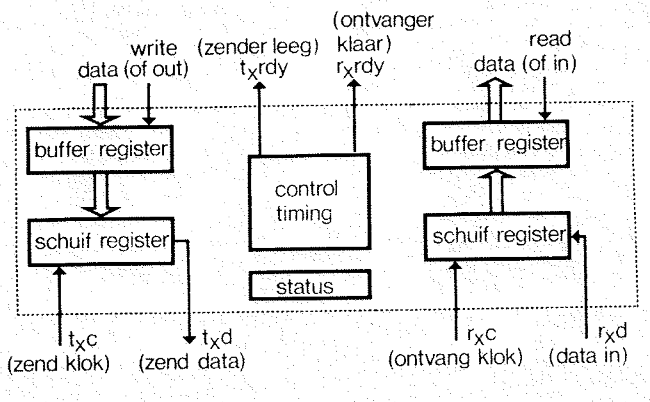


Afb. 14.14. Schema van de zender

dat door de computer gelezen kan worden. Fouten bij de ontvangst worden via foutindicatiebits in het statusregister doorgegeven. De afbeeldingen 14.14, 14.15 en 14.16 laten de opbouw van de zender, de ontvanger en de totale USART zien. Het programma in de computer kan zich nu beperken tot het alleen maar lezen en schrijven naar de register (I/O-poorten of geheugenplaatsen) en heeft verder niets te maken met de complexe lijnprocedure.

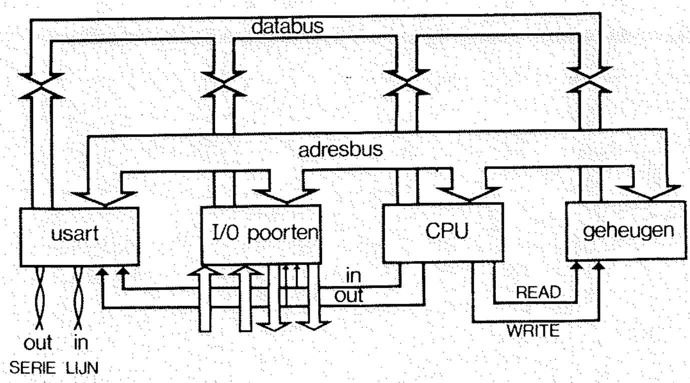


Afb. 14.15. Schema van de ontvanger



Afb. 14.16. De USART

Een randapparaat zal met soortgelijke hardware zijn uitgerust. Daar zullen de lijnsignalen op een zelfde manier worden verwerkt. Het totale schema van een computer, die is uitgerust met een USART vinden we in afbeelding 14.17.



Afb. 14.17. Computer met een USART

Door uitbreidingen door middel van een USART in een microcomputer hebben we nu samen met de normale I/O-poorten de volgende mogelijkheden gekregen:
- Parallelle I/O voor bijvoorbeeld besturing van een of ander proces, een printer e.d.;
- serie I/O naar een terminal of een videodisplay;
- reken- en geheugenmogelijkheden in de computer zelf.

De microcomputer kan voor ons nu een aantal taken uitvoeren, die vroeger alleen maar met complexe schakelingen te realiseren waren.

Samenvatting hoofdstuk 14

In dit hoofdstuk zijn een aantal technieken besproken, die in veel typen microcomputers in een of andere vorm terug te vinden zijn. Er is vooral de nadruk gelegd op de gedachtenwereld, die achter een foutloos I/O-protocol schuilt.

Multiple choice-vragen bij hoofdstuk 14

Deze vragen gaan over de stof van de hoofdstukken 13 en 14.

A. Hoeveel states duren de instructies

```
LXI H, 0F000H
MOV M,C
```

in een 8080 microcomputer samen.

1 3
2 4
3 17
4 20

B. Als we memory-mapped-I/O toepassen, moeten we het I/O poort adres aanbieden als:

1 8 bits geheugenadressen
2 2 bytes I/O-data
3 8 bits adressen voor I/O-poort
4 16 bits adressen voor I/O-poort

C. We passen een geheugen van 62 K bytes toe in een computer. De rest van de 64 K gebruiken we voor memory-mapped-I/O. We krijgen dan maximaal:

1 2 K input- of outputpoorten (elk willekeurig uit deze 2 K te kiezen)
2 1 K inputpoorten en 1 K outputpoorten
3 2 K inputpoorten en 2 K outputpoorten
4 geen enkele I/O-poort

D. a. In een microcomputersysteem mag nooit memory-mapped-I/O en I/O-mapped-I/O tegelijk toegepast worden.
b. In een computer met een I/O-mapped-I/O systeem mogen geen STA <adres> en LDA <adres> instructies worden gebruikt.

1 uitspraak a is goed en uitspraak b is goed
2 uitspraak a is fout en uitspraak b is goed
3 uitspraak a is goed en uitspraak b is fout
4 uitspraak a is fout en uitspraak b is fout

E. Als we bij een computer met een memory-mapped-I/O systeem data inlezen via het volgende programma:

```
LXI H, 0FFFFH
XRA A
ADD M
```

wordt hierbij de parity-flag

1 altijd 0
2 altijd 1
3 aangepast aan de input-data
4 niet veranderd

F. De snelheid, waarmee we opeenvolgende datawoorden bij parallelle output naar buiten kunnen sturen wordt bepaald door:

1 de breedte van de outputpoort
2 de reeks instructies, waarmee we dit doen
3 de plaats van de outputpoort in de I/O-map
4 het aantal bytes, dat we naar buiten willen sturen

G. Een input instructie neemt data over van een inputpoort. Dat wil zeggen:

1 de data gaat van een inputpoort naar het geheugen
2 de data gaat van een inputpoort naar de programmateller
3 de data gaat van een inputpoort naar de accumulator
4 de data gaat van een inputpoort naar de stack

H. De functie van een output latch is:

1 data kortstondig vasthouden gedurende de tijd dat de OUT instructie duurt
2 data langdurig vasthouden
3 data vasthouden tot er weer een willekeurige I/O-instructie komt
4 kijken of er output-data is

I. **a.** Bij memory-mapped-I/O is geen direct transport vanuit register C naar de outputpoort mogelijk
b. Bij input (zonder hand-shake) kan data verloren gaan.

1 uitspraak a is goed en uitspraak b is goed
2 uitspraak a is fout en uitspraak b is goed
3 uitspraak a is goed en uitspraak b is fout
4 uitspraak a is fout en uitspraak b is fout

J. We hebben een systeem met memory-mapped-I/O. Hierbij is het mogelijk om van outputpoort 1000H alleen bit 1 laag te maken met de instructies:

1 LXI H, 1000H
ANI 0FDH
2 LXI H, 1000H
ORI 02H
3 LXI H, 1000H
XRI 02H
4 geen van bovenstaande drie mogelijkheden is correct

Hoofdstuk 15

I/O-programmering

In dit hoofdstuk wordt verder ingegaan op de problemen, die aan de orde komen bij het ontwikkelen van I/O-routines. De parallelle input en output zullen aan de hand van een concrete situatie worden uitgewerkt. We gaan daarna op een aantal zaken in, die we moeten bestuderen, voordat we I/O-driver routines, die beveiligd zijn tegen alle mogelijke fouten die in een systeem kunnen optreden, kunnen ontwerpen. Een I/O driver bestuurt een randapparaat. Deze fouten kunnen lopen van transmissiefouten tot en met het vergeten van het inschakelen van de netspanning bij randapparatuur. De software zal zo ontworpen moeten worden dat met al deze problemen rekening wordt gehouden.

15.1. Input en output

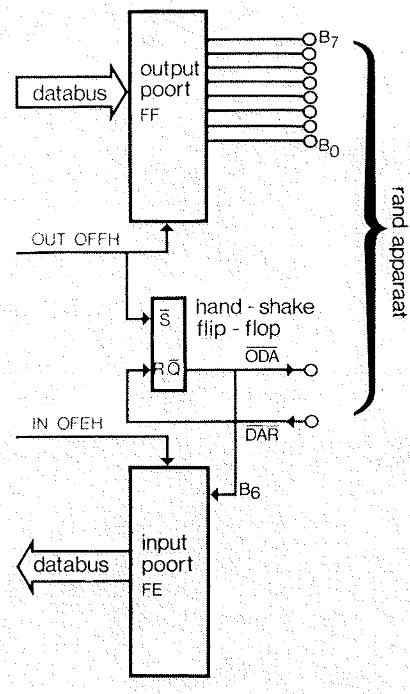
Voordat we een computer echt kunnen toepassen, moeten we ons nog eens realiseren dat de computer niet alleen gebruikt wordt om te rekenen. Reeds in de eerste hoofdstukken zagen we dat een van de belangrijkste functies van een computer bestaat uit het besturen van randapparaten en processen. Het is daarom nodig eens wat dieper op de materie in te gaan die samenhangt met het besturen van randapparaten in situaties, waarbij fouten kunnen optreden. We moeten namelijk in de praktijk er altijd rekening mee houden dat dit kan gebeuren. In deze gevallen moeten we tegen fouten beveiligde programma's hebben, zodat we toch door kunnen gaan. Meestal is het niet te vermijden dat hardware fouten maakt zeker als het om mechanische apparaten gaat. In deze situaties mogen we nooit de data verliezen, en moeten we een teruggang van de prestaties zoveel mogelijk voorkomen.

15.2. Parallelle output op handshake basis

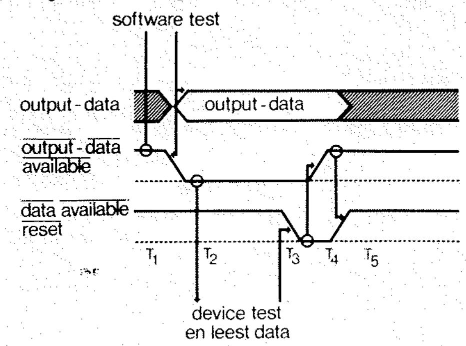
In het vorige hoofdstuk hebben we uitgebreid besproken hoe parallelle I/O bij een microcomputer werkt en hoe de interface-specificaties tussen twee systemen kunnen zijn. Nu zullen we deze theorie eens toepassen in een microcomputersysteem. Als voorbeeld hiervoor hebben we gekozen voor de parallelle input- en outputstructuur van de EXIDY SORCERER personal computer. Deze personal computer bevat een output-poort van 8 bit breed en een input-poort van 8 bit breed. We hebben hierbij te maken met I/O-mapped I/O.

Parallelle output-data: output-poort FF bit 0 t/m 7
Parallelle input-data: input-poort FF bit 0 t/m 7
Handshake signalen: input-poort FE
output-data available bit 6
input-data available bit 7
Externe signalen: (waar de computer niet rechtstreeks bij kan)
data available reset
input-data accepted

De output-poort kan met de OUT FF instructie en de input-poort met de IN FF instructie geadresseerd worden. Behalve de datapoorten bevat de parallelle interface ook nog een viertal extra signaallijnen voor de zogenaamde handshake (zie afbeelding 15.1).
Bij de output is er een output-data available lijn, die vanuit de microcomputer naar buiten gaat. Met deze lijn geven we aan de buitenwereld te kennen dat er een nieuw datawoord beschikbaar is in het output-dataregister (hier output-poort FF). Wanneer we data naar dit register schrijven, wordt tevens de handshake flip-flop geset (dat wil zeggen hoog gemaakt). Dit gebeurt automatisch door de hardware in de Sorcerer. De output-data available is aangesloten op de inverse uitgang van de flip-flop en zal dus hierbij laag worden. Dit betekent voor het externe apparaat, dat wanneer de output-data available lijn laag wordt, er nieuwe data voor overname in de output-poort geschreven is. Wanneer nu door het externe device deze data is gelezen, zal het externe device een zogenaamd data available reset signaal moeten geven. Hiermee wordt door de Sorcerer de handshake flip-flop gereset (dus 0 gemaakt). Daardoor wordt ook de inverse uitgang (output-data available) weer hoog gemaakt. Deze output-data available lijn kan door de computer worden gelezen. De lijn is namelijk aangesloten op bit nummer 6 van de input-poort FE. Hierdoor is het mogelijk dat de computer test of het externe device de data heeft overgenomen. We zien hiermee dat het protocol zoals dat in de vorige hoofdstukken is beschreven, dus helemaal netjes kan worden toegepast. In afbeelding 15.1 zien we het schema van deze output-structuur, die op hand-shake basis werkt. Een ander soort afbeeldingen, die vaak worden gebruikt bij het beschrijven van interfaces, zijn de zogenaamde timing-diagrammen. Een timing-diagram geeft het verloop van de signalen weer als functie van de tijd.



Afb. 15.1. Output-structuur



Afb. 15.2. Timing-diagram voor output

De tijd staat bij de horizontale as en signaalniveaus staan bij de verticale as. In het timing-diagram van afbeelding 15.2 zien we hoe de signalen bij de output-acties verlopen. In de afbeelding zien we van boven naar beneden achtereenvolgens de datalijnen, die in een

groep van 8 zijn gebundeld. Omdat deze datalijnen elke willekeurige waarde kunnen aannemen, zijn ze door een tweetal lijnen weergegeven. De ene lijn is laag en de andere hoog. Op de tijdstippen dat er signaalwisselingen optreden, dus zodra nieuwe data naar buiten worden gestuurd, wisselen de lijnen van niveau. Daaronder is in het timing-diagram het signaal output-data available getekend. Door een streep te zetten boven de tekst output-data available, geven we aan dat het signaal laag actief is. Dit betekent dat de output-data geldig is, wanneer dit signaal laag is. De onderste lijn in het diagram is het data available reset signaal. Dit is een terugmeldingssignaal van het externe device naar de computer. Zodra dit signaal laag wordt, betekent dit dat het output-device de data heeft overgenomen. Als gevolg hiervan zal het output-data available signaal in-actief, dus hoog worden. Na het laag worden van de data available reset-lijn wordt deze ook weer hoog.
We zullen nu afbeelding 15.2 nog eens nader bekijken. Op het tijdstip T1 hebben we een situatie, waarbij zowel output-data available als data available reset beide hoog zijn. Dit is als uitgangspositie te beschouwen. De computer kan het output-data available signaal testen door bit 6 van input-poort FE te bekijken. Indien output-data available hoog is, betekent dit dat het output-register niet gevuld is en er dus door de computer nieuwe data ingeschreven mogen worden. Dit zal gebeuren door data naar output-poort FF te schrijven. Tegelijk met het inschrijven van de data in de output-poort zal de handshake flip-flop geset worden. Als gevolg hiervan zal output-data available laag worden. Deze overgang is een direct gevolg van de OUT FF instructie. We zien in de bovenste lijnen dat er een signaalwisseling plaatsvindt van het gearceerde stuk naar het stuk waarin de tekst OUTPUT DATA staat. In dit laatste deel is de data geldig. De data out available lijn dient nu als flag voor de buitenwereld. Zodra het externe apparaat (of bijvoorbeeld de andere computer, waarmee we zouden kunnen communiceren) de data heeft gelezen, zal deze hierop een data available reset signaal geven dat korte tijd laag gemaakt wordt. In de afbeelding is dit het tijdstip T3. Als gevolg hiervan zal de handshake flipflop gereset worden en dus output-data available weer hoog worden. Het betekent dan ook dat de output-data, die weliswaar in het output-register blijven staan, niet meer mogen worden gebruikt. Ze zijn een keer overgenomen en output-data available is niet meer actief (ofwel de data is in feite niet meer geldig). We zitten hierbij op tijdstip T4.
Nadat het resetsignaal weg is (op tijdstip T5) bevinden we ons weer in de uitgangssituatie. In het timing-diagram is door middel van enkele pijlen oorzaak en gevolg van signaalovergangen aangegeven.

In het programma van afbeelding 15.3 zien we de acties nog een keer duidelijk weergegeven. Het programma is in assembleertaal geschreven. Hier zijn instructies van de 8080 gebruikt. Het programma zorgt voor de output van een byte. Deze byte staat in het register A (de accumulator), zodra het programma wordt aangeroepen. We moeten dit register dus vullen met de naar buiten te sturen data net voordat we het programma aanroepen. We hebben aan het programma de naam PAROUT gegeven. Het kan dus worden aangeroepen vanuit een ander programma als een subroutine en wel met de instructie CALL PAROUT. De assembler zal zelf het adres van

PAROUT invullen als we de routine assembleren. Als we het programma nader bestuderen, zien we dat we beginnen met een PUSH PSW instructie. Deze instructie zorgt ervoor dat de waarde van register A op de stack wordt gezet. Dit doen we om te zorgen dat bij het testen van de handshake lijn (wat noodzakelijk is voor de juiste interacties en gebeurt via het A-register) de data in het A-register niet worden verminkt. We zouden dan geen correcte data meer ter beschikking hebben. De volgende instructie is een input-instructie, waarbij de input-poort FE wordt ingelezen. Dit is de poort, waarop we het output-data available signaal hebben aangesloten.

```
PAROUT:   PUSH   PSW       ; SAVE DATA
PAROT1:   IN     OFEH      ; READ CONTROL
          ANI    40H       ; SELECT BIT 6
          JZ     PAROT1    ; NON-ZERO, GO ON
          POP    PSW       ; LOAD DATA
          OUT    OFFH      ; DATA OUT
          RET
```

Afb. 15.3. Output-routine

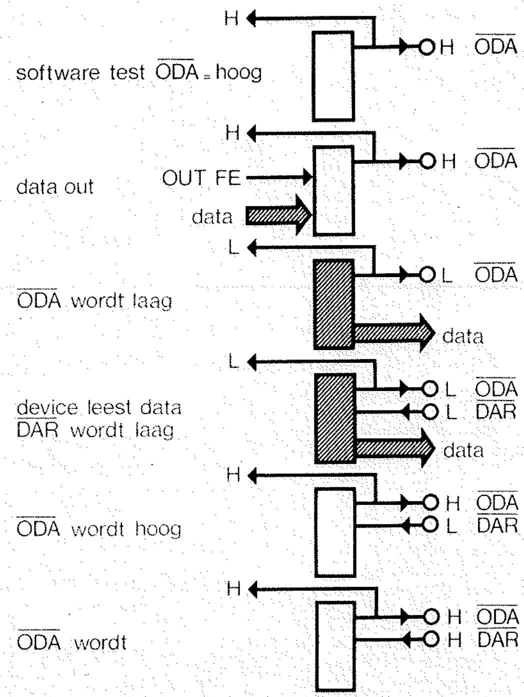
Door nu te testen of bit 6 hoog is, kunnen we zien of er nog oude data in de output poort staan, of dat deze data reeds zijn overgenomen en de poort weer beschikbaar is voor nieuwe output. Als het register leeg is, dus bit 6 is niet 0, gaan we door. Is het register nog vol, dan zal bit 6 gelijk aan 0 zijn, dus is de vorige data nog niet overgenomen door het apparaat. We springen dan terug naar de tweede instructie van deze routine: het label PAROT1. Indien we doorgaan moeten we de waarde van A, die we zojuist op de stack hebben gezet, terughalen. Dit doen we met de POP PSW instructie. Daarna staat de data weer in het A-register en kunnen we via een OUT FF instructie de waarde naar output-poort FF schrijven. Hiermee is de routine klaar.

Wanneer we de relatie tussen het programma en het timing-diagram bekijken, zien we dat we bij het begin van het programma in situatie T1 zitten. We voeren dan de test op bit 6 uit. Er wordt naar T2 overgegaan als we de data naar buiten zenden. De verdere afwikkeling voor situatie T3 en T4 vindt helemaal plaats door externe signalen en is dus door de software niet te beïnvloeden. Het zal duidelijk zijn dat door het boven beschreven interface-programma en de hardware interface een compleet fail-systeem is gerealiseerd. Door de flip-flop wordt namelijk onthouden of het data-buffer leeg of vol is.

Soms worden in computersystemen andere namen gebruikt voor output-data available en data available reset. Voor output-data available zien we vaak de benaming OUTPUT BUFFER FULL. Deze naam is veel beter, omdat de betekenis van het signaal veel beter door de naam wordt gedekt. In plaats van data available reset, wordt ook soms de benaming ACKNOWLEDGE gebruikt. ACKNOWLEDGE betekent zoiets als terugmelding of bevestiging.

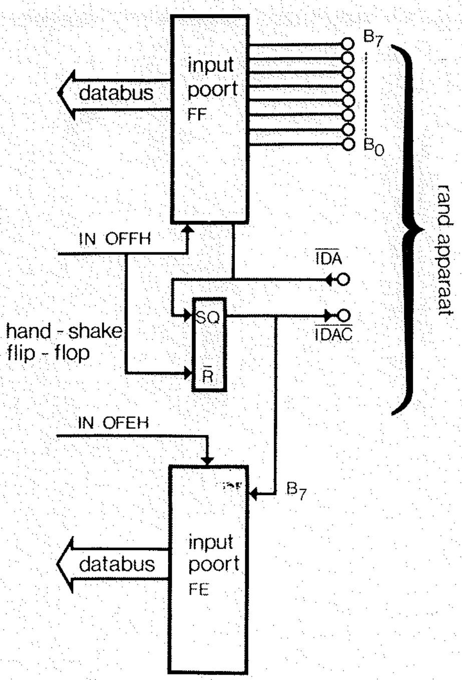
In afbeelding 15.4 is het handshake spel nog eens op een andere wijze symbolisch weergegeven. Probeer zelf de verschillende fasen van de output-acties te herkennen.

In de hierboven beschreven interface vinden we een groot deel van de denktrant terug, die in het vorige hoofdstuk is beschreven. Het is nuttig om dit en het vorige hoofdstuk nog eens grondig met elkaar te vergelijken.



Afb. 15.4. Symbolische handshake, output

De acties, die achtereenvolgens worden uitgevoerd zijn:
- De CPU test op data output available.
- Als dit signaal hoog is mag data naar buiten worden gestuurd.
- De data worden in de poort gezet en outputdata available wordt laag.
- De externe apparatuur leest de data.
- Het externe apparaat maakt data available reset tijdelijk laag.
- Output-data available wordt hierop hoog.



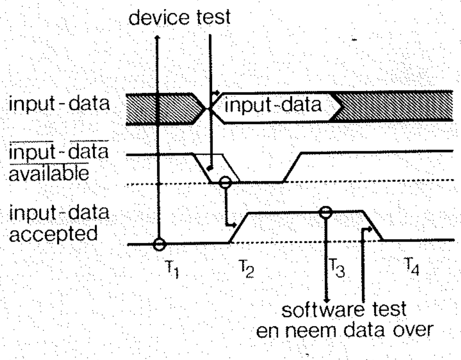
Afb. 15.5. Input-structuur

15.3. Parallelle input op handshake basis

Evenals bij de parallelle output hebben we bij de parallelle input een systeem met een handshake flipflop. Het systeem werkt op ongeveer gelijke wijze (zie afbeelding 15.5 voor de hardware structuur).

We hebben een input-bufferregister, dat de mogelijkheid bezit om data vast te houden. Dit in tegenstelling tot een normale input-poort. Deze kan namelijk geen data vasthouden, maar alleen kortstondig doorlaten. Dit bufferregister is tevens als input-poort FF te lezen. De buffer kunnen we met data vullen door de data aan te bieden en daarbij tevens een signaal input-data available te geven. Dit input-data available signaal zal een laaggaande puls zijn. Dit laaggaande signaal geeft een laadpuls aan het input-buffer en zal tevens de handshake flip-flop setten. Dit heeft tot direct gevolg dat input-data accepted hoog wordt, want deze lijn is direct aan de uitgang van

flipflop gekoppeld. De buitenwereld (het device) ziet hieraan meteen dat het bufferregister gevuld is en kan dus de externe data wegnemen en de puls, die deze data begeleidt, tevens laten vervallen (dus hoog maken). De uitgang van de flipflop is ook gekoppeld aan een input-poort voor besturing. Dit is dezelfde input-poort, die we bij de handshake van de output-acties gebruiken. Nu testen we echter op bit 7. Aan dit bit kan de software zien of er nieuwe data beschikbaar is in het input-buffer. In de software kunnen we weer op dit bit testen en afhankelijk van de waarde van dit bit de data overnemen. Het overnemen van de data gebeurt met behulp van een IN FF instructie. Hierbij wordt het input-buffer uitgelezen (dus op de databus gezet) en tevens wordt de handshake flipflop gereset. Hierdoor zal het input-data accepted signaal laag worden. We moeten er hierbij vooral voor zorgen, dat de input-data available puls altijd korter is dan de reactietijd van de software. Want een flipflop, die we willen setten en resetten (dus het lezen van het register en het tegelijkertijd inklokken van het register) zal fouten opleveren. We zijn nu weer in de uitgangstoestand terechtgekomen. In deze toestand is zowel input-data available als input-data accepted in de oude toestand (rusttoestand), dus respectievelijk hoog en laag terechtgekomen. De hele cyclus is nu dus weer opnieuw klaar voor een input-actie. Beide partners weten precies wat er gebeurd is en de verantwoordelijkheden voor de interface zijn exact vastgelegd.



Afb. 15.6. Timing-diagram voor input

In afbeelding 15.6 vinden we het timing-diagram voor input op handshake basis. Het is bijna identiek aan het schema voor output. Hier verlopen de acties van inklokken en testen echter net in tegenovergestelde volgorde. Eerst zal hier de hardware het externe device testen en data inklokken en daarna neemt de interne software de test over. We gaan nu niet verder op het timing-diagram in.

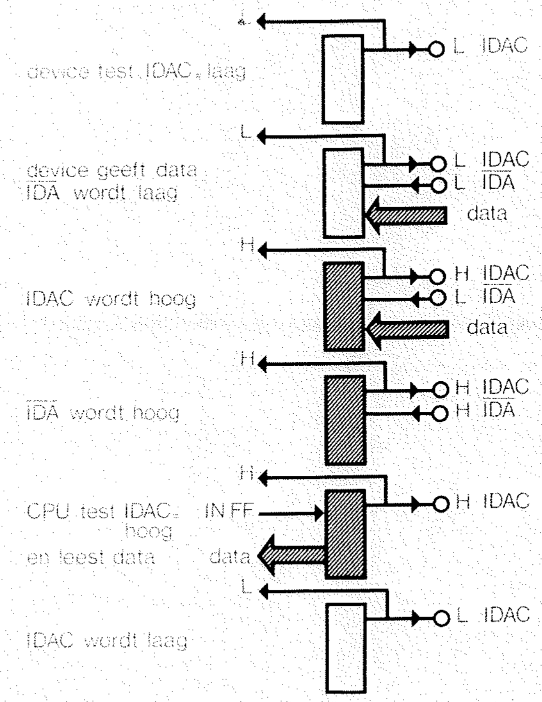
Als we het programma voor input bekijken (afbeelding 15.7) zien we een grote overeenkomst met het programma voor de output. In eerste instantie zullen de acties door de hardware worden ondernomen,

```
PARIN:    IN     OFEH      ; READ CONTROL
          ANI    80 H      ; SELECT BIT 7
          JZ     PARIN     ; NON-ZERO, GO ON
          IN     OFFH      ; READ DATA
          RET
```

Afb. 15.7. Input-routine

namelijk het inlezen van de data. Daarna zal de software kunnen testen op het actief worden van input-data accepted. Als deze lijn hoog is, betekent dit voor de software dat er een nieuw byte in het input-buffer beschikbaar is. Dus in de eerste instructies zullen we de besturings-input-poort lezen en daarvan bit 7 selecteren. Als dit niet hoog is, zullen we dit stuk van het programma voortdurend blijven uitvoeren. Zodra het bit wel hoog wordt, betekent dit dat er nieuwe data beschikbaar is en kan deze worden overgenomen via de IN FF instructie. De data zal nu in het A-register staan en we kunnen uit de routine terugkeren met een returninstructie. Hierbij wordt teruggekeerd naar de aanroepende routine. De software zal dus ergens in T3 van het timings-diagram ingrijpen. We moeten vooral letten op sluitende afspraken. Als we de input-data available puls kort nemen, zal het hele interface-afspraken fail-safe werken, omdat de software relatief langer zal duren dat het input-data available signaal laag zal zijn. Beide partners weten hier weer exact wat elkaars toestand zal zijn.

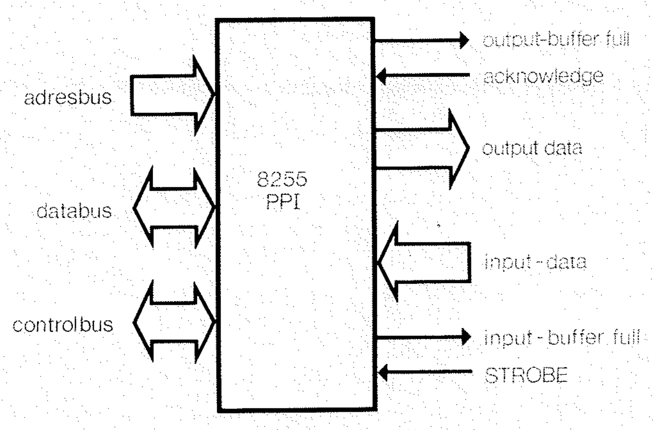
Net als voor output kunnen we ook voor input een overzicht maken van de uit te voeren stappen. Zie hiervoor ook afbeelding 15.8 waarin een en ander schematisch is weergegeven.



Afb. 15.8. Schematisch input handshake

De acties zijn achtereenvolgens:
- Het randapparaat test of het input-buffer leeg is.
- Zo ja, dan mag er data worden aangeboden die met behulp van het input-data available signaal wordt ingeklokt in het buffer.
- In de handshake hardware slaat de input-data accepted lijn om.
- De CPU test op de input-data accepted lijn om te zien of er soms nieuwe data beschikbaar is.
- Als er data is wordt de input-poort uitgelezen en zal het input-data accepted signaal laag worden gemaakt.

In andere computersystemen worden ook weer andere namen gebruikt. Het input-data available signaal wordt soms STROBE signaal genoemd en het input-data accepted signaal heet in veel systemen INPUT BUFFER FULL. Ook hier hebben we meer sprekende namen dan in het hierboven beschreven geval. STROBE is namelijk een kloksignaal, dat de input-data begeleidt. INPUT BUFFER FULL is een duidelijke benaming, die aangeeft dat er nieuwe data in het input-buffer staan. Als we verder kijken dan de EXIDY SORCERER personal computer, zullen we vaak I/O-systemen ontdekken, waarbij deze INPUT BUFFER FULL, OUTPUT BUFFER FULL, ACKNOWLEDGE en STROBE signalen in een chip geïntegreerd zijn. Een voorbeeld hiervan zijn de parallel peripheral interface bouwstenen van de grote halfgeleider fabrikanten. Ze werken allemaal op een bijna gelijksoortige wijze. Een voorbeeld hiervan zien we nog eens terug in afbeelding 15.9. Hierbij is een parallelle bouwsteen aangegeven, die al jaren lang gebruikt wordt in diverse microcomputersystemen. Het is de zogenaamde 8255 PPI van INTEL. Hierbij is de werkmode getekend, waarbij we een input- en een output-buffer hebben op zogenaamde handshake I/O-basis. Dit betekent dat we een INPUT BUFFER FULL en een STROBE signaal samen hebben voor input en een OUTPUT BUFFER FULL samen met een ACKNOWLEDGE signaal voor output. Er zijn veel varianten van deze bouwstenen in gebruik. Het zou nuttig zijn om zelf eens dieper op deze materie in te gaan door de fabrieksdocumentatie van bijvoorbeeld INTEL, MOTOROLA, SIGNETICS, FAIRCHILD e.d. te bestuderen.



Afb. 15.9. Universal parallel interface chip

15.4. I/O-driver routines

De routines, die deze besturing voor externe apparatuur voor hun rekening nemen, worden I/O-driver routines genoemd. Zo'n routine zal zorgen dat op de juiste wijze data worden uitgewisseld met het randapparaat (of proces). Dit zagen we reeds in het vorige deel van dit hoofdstuk.

Wat heeft zo'n I/O-driver nu nog meer voor functies?

De I/O-driver zal uit een ander programma data accepteren. In dit hogere programma worden de data geproduceerd, die bijvoorbeeld naar buiten moeten. Deze data zullen dan door de I/O-driver op de

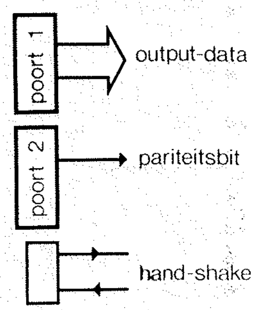
juiste wijze aan het randapparaat moeten worden aangeboden. Dat wil zeggen dat de driver-routine moet testen of het randapparaat klaar is om data te ontvangen. Zo niet, dan zal de driver-routine telkens weer moeten testen tot het randapparaat klaar is voor deze actie. Mocht dit echter te lang duren, dan zal de driver-routine aan de hem aanroepende hoofd-routine terug moeten melden dat er een fouttoestand bestaat. De hoofdroutine kan dan bijvoorbeeld een operator waarschuwen, omdat er iets mis is. Dit testen op een te lange tijdsduur houdt in, dat er in de I/O-driver routine een deel opgenomen moet zijn, dat aftelt hoe vaak de status van het I/O-device wordt getest. Afhankelijk van de complexiteit van deze tests kunnen wachtroutines worden ingebouwd met behulp van een telalgorithme. Als een bepaalde tellerstand wordt overschreden, betekent dit voor de gebruiker dat de I/O-driver routine er niet in is geslaagd in een door ons opgegeven tijd de data op een verantwoorde wijze over te dragen.

```
PAROUT:    PUSH   PSW       ; SAVE DATA
           PUSH   B         ; SAVE B, C
           LXI    B, 1000   ;
PAROT 1:   IN     0FEH      ; READ CONTROL
           ANI    40 H      ; SELECT BIT 6
           JNZ    READY     ; NON-ZERO, GO TO READY
           DCX    B
           MOV    A, B      ; COUNT
           ORA    C
           JNZ    PAROT1    ; WAIT
           POP    B         ; LOAD B, C
           POP    PSW
           XRA    A
           JMP    RETURN    ; Z-FLAG = 1, RETURN
READY:     POP    B         ; LOAD B, C
           POP    PSW       ; LOAD DATA
           OUT    0FFH      ;
           XRA    A         ;
           INR    A         ; Z-FLAG = O
RETURN:    RET
```

Afb. 15.10. Aanpassing output-routine

We zouden in het programma-voorbeeld van afbeelding 15.3 in de test op de statuslijn een teller kunnen opnemen (zie hiervoor afbeelding 15.10). Hierin wordt via het registerpaar B, C geteld hoe vaak we door de lus gaan. Wanneer dit te lang duurt, keren we terug naar het aanroepende programma. We kunnen hierbij door middel van bijvoorbeeld de Z-flag meedelen of de output goed is gegaan, ofwel is afgebroken als gevolg van een time-out. We kunnen hiervoor de Z-flag respectievelijk op 0 of 1 zetten. In de testlus worden nu wel de registers B en C verminkt. Dit houdt wel in dat we eerst B en C op de stack moeten redden (via een PUSH B instructie) en later weer moeten herstellen (met een POP B instructie). Bij een fout zal de aanroeper van de I/O-driver routine zelf de verantwoordelijkheid terugkrijgen over data en de nodige acties moeten ondernemen. Bij de terugkeer moet de aanroeper testen op de waarde van de Z-flag.

Hieruit blijkt reeds dat een I/O-driver routine veel meer inhoudt dan een simpele routine, die een byte vanuit de computer overdraagt naar bijvoorbeeld de printer. In de I/O-driver routine moet ten eerste het handshake protocol van het randapparaat worden gerealiseerd. Verder moet er bovendien een zogenaamde time-out bewaking zijn, zodat de routine zich niet onnodig 'ophangt', indien bijvoorbeeld een printer niet aan staat. We zouden dan namelijk oneindig lang blijven wachten tot er iets gebeurt en er gebeurt niets. Als we bovendien merken dat de computer zich op deze wijze in een loop bevindt en we zouden toch het randapparaat inschakelen, kan door de overgangen op de lijnen van het randapparaat er bijvoorbeeld een byte van de data verloren gaan. Dit is ook niet gewenst. Een ander probleem, dat we bij I/O-routines aantreffen is de foutbewaking. Een randapparaat zal namelijk, nadat er een byte is geaccepteerd, dit controleren op fouten en daarna terugmelden of een en ander goed is verlopen. Afbeelding 15.11 geeft de extra pariteitslijn als negende bit op een andere output-poort. We zullen eerst dit pariteitsbit moeten uitsturen en dan de data samen met het output-data available signaal moeten geven. Zo zal er dan bijvoorbeeld worden getest op de pariteitslijn. Er zal ook kunnen worden getest of de mechanische acties, die door het randapparaat uitgevoerd moeten worden, helemaal correct zijn verlopen of er zal teruggemeld worden dat het buffer in de elektronika van het randapparaat nog meerdere bytes kan ontvangen, enz.



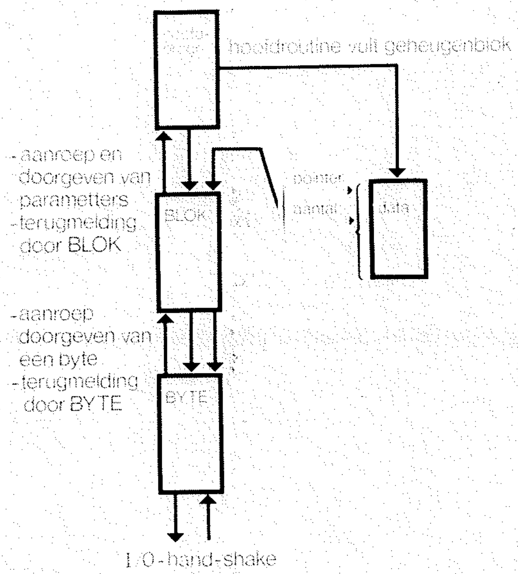
Afb. 15.11. Output met pariteitsbit

15.5. Universele I/O-drivers

Vaak zullen we een I/O-driver routine niet voor een bepaalde toepassing alleen schrijven, maar er zullen meerdere varianten in worden behandeld. Dit wil zeggen dat de I/O-driver bijvoorbeeld geschikt moet zijn voor meerdere typen van een bepaald I/O-device. Als we dit niet zouden doen betekent dit dat we de ene keer te maken hebben met de routine met een bepaalde set van mogelijkheden en misschien een half uur later weer met een soortgelijke routine met slechts enkele andere mogelijkheden. Hierbij zal toch altijd weer een deel van de eerste routine aanwezig zijn. Als we deze gedachte voortzetten, zien we direct dat de verschillende I/O-drivers heel veel dezelfde deelroutines hebben, maar op slechts kleine punten verschillend zijn. Bij een algorithme voor serie-datatransport zal in het ene geval gebruik worden gemaakt van een even pariteit, de andere keer van een oneven pariteit, dan weer eens van een pariteits-bit altijd 1, of altijd 0. Dit zou betekenen, dat we voor elke variant een ander programma in de machine nodig hebben. Maar het serie-algorithme is afgezien van de pariteitsbit voor al deze routines identiek (zie hoofdstuk 14). Er zouden slechts kleine delen van de routine moeten worden veranderd. Een tweede variant, die we kunnen bedenken, is het aantal stopbits dat wordt toegepast. We kunnen een routine hebben voor serie-transmissie met één stopbits, met twee stopbits en er zijn zelfs computers, die anderhalve bittijd voor de stop nemen. Als we een en ander op een rijtje zetten kunnen we concluderen dat het nuttig is om een algemene driver voor serie-transmissie te schrijven, waarbij we eerst aangeven hoe de bepaalde settings van bijvoorbeeld pariteit en aantal stopbits zijn. Deze gedachte vinden we in de meeste goede I/O-drivers terug.

15.6. Bloktransport

Een ander probleem, dat bij I/O-driver routines voorkomt is het aantal bytes, dat we willen overdragen. Het zou mooi zijn als we een routine hebben, waaraan we opgeven (bijvoorbeeld via een wijzer (pointer)), waar ons datagebied zich in het geheugen bevindt. Behalve de pointer moeten we ook nog aangeven hoe lang ons datablok zal zijn (dus hoeveel bytes we willen overdragen). Deze twee gegevens, dus het begin van ons datagebied en de lengte, zijn een input (een beter woord zou zijn parameters) voor een programma, die op zijn beurt weer een ander programma zal aanroepen, bijvoorbeeld weer onze serie-transmissie-routine. We noemen het aanroepende programma wel een hogere orde programma. Het aangeroepen programma noemen we dan een lagere orde programma. De computer kan dan deze laatste routine aanroepen met als parameter die pointer en de bloklengte. Het datatransport zal byte voor byte gebeuren, zonder dat we ons om de rest van de afwikkeling hoeven te bekommeren. We spreken nu van een schil op het hoger niveau (zie hiervoor afbeelding 15.12).



Afb. 15.12. Niveaus van I/O-routines

Een hoofdroutine zal op een of andere wijze een aantal characters (bytes) genereren, die op een printer moeten worden afgedrukt. Als bijvoorbeeld een volledige regel beschikbaar is, zullen we een hulproutine (BLOK) aanroepen. Deze routine moet ervoor zorgen, dat de regel byte voor byte via een tweede (nog dichter bij de hardware staande) routine aan de printer wordt doorgegeven. Het laagste niveau (BYTE) bestuurt direct de I/O-lijnen. We komen nu echter op een ander probleem. We moeten aangeven dat we een blokdata willen overdragen naar een device. Daarbij moet de lengte en het begin worden opgegeven. Er kan echter om een of andere reden onderweg (tijdens het uitzenden van een byte) iets mis gaan. Als er iets mis is gegaan, moet dit aan de hogere orde routine worden meegedeeld.

Deze moet dan precies weten hoeveel bytes er correct zijn overgekomen en wat er is misgegaan. Hiervoor geven we behalve ons datablok en de lengte nog een derde parameter mee aan de lagere orde I/O-device, namelijk een adres waar we naar toe springen, zodra er iets is misgegaan. Dit is een ERROR handling pointer. Als er een fout optreedt, zal er via die hogere orde routine op een bepaald communicatiegebied (weer enkele plaatsen, die we in het geheugen reserveren) een code worden gezet. Deze code geeft aan, dat de transmissie goed beëindigd is ofwel op byte nummer K is afgebroken en dat daarbij foutnummer N is opgetreden. De aanroepende routine weet dan bij het beëindigen van het datatransport precies hoe een en ander is verlopen. We weten dan precies wat er na het herstellen van de fout nog overgezonden zal moeten worden. Vaak wordt deze foutbewaking en het afhandelen van allerlei foutroutines onderschat en vergeten. Dit is een van de redenen dat microprocessorsystemen wel eens een slechter resultaat hebben als gewone logica. Dit betekent niet dat de microprocessorsystemen slechter zijn, maar het betekent wel dat de gebruiker niet nagedacht heeft bij het realiseren van zijn werk.

Samenvatting hoofdstuk 15

In dit hoofdstuk is vrij diepgaand besproken wat de aspecten zijn, die bij het programmeren van I/O-drivers niet vergeten mogen worden. Het zal een aanzet moeten zijn om zelf interfaceproblemen aan te pakken.

Multiple choice-vragen bij hoofdstuk 15

Deze vragen gaan over de stof van de hoofdstukken 14 en 15

A. In een programma voor hand-shake input wordt door het randapparaat eerder de status: data geldig gegeven dan de data

1 dit mag nooit
2 dit mag wel, want de computer is toch niet snel
3 dit mag wel bij zeer snelle randapparaten
4 dit kan in extra wachttijd resulteren

B. Bij een serie data-transportsnelheid wil 110 bits per seconde zeggen:

1 een bittijd van 9,1 milliseconden
2 een bittijd van 110 milliseconden
3 110 woorden per seconde
4 1 woord per 110 seconden

C. Wat is fout?
De functie van een startbit bij serie I/O is:

1 herkenning van het begin van een nieuw woord mogelijk te maken
2 de bepaling van de baud-rate te starten
3 woordsynchronisatie te starten
4 onderscheid met de rusttoestand te maken

D. a. Serie transmissie heeft geen directe terugmelding.
b. Serie transmissie m.b.v. een USART geeft veel minder software overhead dan geprogrammeerde serie I/O zonder USART.

1 uitspraak a is goed en uitspraak b is goed
2 uitspraak a is fout en uitspraak b is goed
3 uitspraak a is goed en uitspraak b is fout
4 uitspraak a is fout en uitspraak b is fout

E. a. Een USART controleert het aantal stopbits bij het ontvangen van data.
b. Een USART controleert de pariteit bij het ontvangen van data

1 uitspraak a is goed en uitspraak b is goed
2 uitspraak a is fout en uitspraak b is goed
3 uitspraak a is goed en uitspraak b is fout
4 uitspraak a is fout en uitspraak b is fout

F. a. Bij input (volgens afb. 15.5), dient de hand-shake flip-flop om aan de processor door te geven dat er nieuwe data is ingeklokt in de inputbuffer.
b. Als we bij input via het hand-shake-protocol werken kan de input-data slechts een keer gelezen worden.

1 uitspraak a is goed en uitspraak b is goed
2 uitspraak a is fout en uitspraak b is goed
3 uitspraak a is goed en uitspraak b is fout
4 uitspraak a is fout en uitspraak b is fout

G. **a.** Bij hand-shake output (volgens afb. 15.1) is het niet mogelijk om, als de poort gevuld is met data, er weer andere data in te schrijven, zolang het I/O-device geen data available reset heeft gegeven.
b. Bij gebufferde input vult het input-device de input-buffer met data.

1 uitspraak a is goed en uitspraak b is goed
2 uitspraak a is fout en uitspraak b is goed
3 uitspraak a is goed en uitspraak b is fout
4 uitspraak a is fout en uitspraak b is fout

H. **a.** In het programma van afb. 15.7 mogen we
JZ PARIN
vervangen door:
JM PARIN
b. In het programma van afb. 15.7 mogen we:
ANI 80 H
JZ PARIN
vervangen door:
RAL
JNC PARIN

2 uitspraak a is goed en uitspraak b is goed
2 uitspraak a is fout en uitspraak b is goed
3 uitspraak a is goed en uitspraak b is fout
4 uitspraak a is fout en uitspraak b is fout

I. **a.** Een time-out bewaking voorkomt 'hang up'.
b. Transmissie foutbewaking kan gerealiseerd worden d.m.v. pariteitscontrole.

1 uitspraak a is goed en uitspraak b is goed
2 uitspraak a is fout en uitspraak b is goed
3 uitspraak a is goed en uitspraak b is fout
4 uitspraak a is fout en uitspraak b is fout

J. We sturen een pariteitsbit (als 9de bit) uit via een bit 0 van poort 0FAH. De data sturen we uit via poort 0FBH.
Welke uitspraak is juist?

1 de pariteitsbit moet aan de data voorafgaan
2 de pariteitsbit moet na de data komen
3 we hebben een databus van 9 bits nodig
4 de hand-shake werkt niet meer

Hoofdstuk 16

Het monitorprogramma

In dit hoofdstuk wordt het zogenaamde monitorprogramma besproken. Met behulp van dit programma is het mogelijk om op machinetaalniveau geheugenplaatsen uit te lezen; de inhoud van geheugenplaatsen te veranderen; programma's te starten; geheugenblokken op cassetteband te schrijven en data van cassetteband te lezen. De meeste personal computers bevatten een monitorprogramma. Deze programma's zijn alle gelijk van structuur. Alleen de commando's zullen vaak uit andere letterreeksen bestaan. Hier zullen aan de hand van de Teleac/DAI personal computer de mogelijkheden van een monitor worden behandeld, omdat deze zeer uitgebreide mogelijkheden geeft. Een gebruiker zal ze dan ook voor elke andere computer kunnen begrijpen. Het effect van de monitorcommando's zal aan de hand van een reeks voorbeelden worden uitgewerkt.

16.1. Het monitorprogramma (Utility Package)

Naast de gebruikelijke BASIC-interpreter bevatten de meeste personal computers een zogenaamd monitorprogramma. Dit wordt vaak *utility software package* genoemd. Dit is een programma, dat speciaal is bedoeld om de gebruiker via het toetsenbord in machinecode (d.w.z. op hexadecimaal niveau) programma's te testen en te verbeteren. Het monitorprogramma geeft ons de mogelijkheid om direct in het geheugen van de computer in te grijpen. Dit betekent dat we de inhoud van geheugenplaatsen kunnen bekijken en veranderen. Het is ook mogelijk om de inhoud van de registers van de CPU te bekijken en te veranderen. Ook biedt het Package ons de mogelijkheid om tijdens de uitvoer van een programma dynamisch (stap voor stap) te bekijken wat er zoal in de registers gebeurt. We kunnen daaruit dan afleiden of de acties naar wens gebeuren, of waarom de acties niet het effect, dat we ons hadden voorgesteld, laten zien. Naast deze geheugen- en CPU-manipulaties is het mogelijk om met behulp van het Utility Package ook op hexadecimaal niveau programmadelen (dus geheugeninhouden) naar een cassetteband weg te schrijven. De data, die op de band zijn opgeslagen, kunnen dan later weer in het geheugen worden geladen en als programma worden uitgevoerd.
Het programmeren op hexadecimaal niveau (dit wil zeggen op machinecode niveau) biedt ons de mogelijkheid om de snelheid van executie van programma's helemaal in eigen hand te houden. Als we namelijk een programma zouden uitvoeren met behulp van BASIC-statements, weten we nooit precies hoe lang de executie van het

programma duurt. Wanneer we echter machinecode gebruiken, ligt van tevoren exact vast hoe lang een bepaalde instructie duurt. We kunnen dan ook uitrekenen wat de tijdsduur van een bepaald programma zal zijn. De tijdsduur voor elke machine-instructie staat namelijk vast, zie hoofdstuk 12. We moeten er echter wel voor waken dat we niet te veel op deze methodiek gaan bouwen. Het programmeren in machinecode via een monitorprogramma is namelijk geen gemakkelijke manier van programmeren. We moeten hier van elke uit te voeren instructie vanuit de mnemonics de bijbehorende hexadecimale machinecode opzoeken. We moeten daarbij zelf de adressen, die in het programma worden gebruikt, berekenen. Indien het programma niet correct werkt, is het daardoor erg moeilijk om later modificaties aan te brengen. Wanneer we de beschikking hebben over een assemblerprogramma, zijn veel van deze moeilijkheden niet aanwezig. De assembler vertaalt dan van symbolische namen naar hexadecimale waarden voor ons. Het is echter toch wel nuttig om eens een paar programma's in machinecode te schrijven en deze te testen. Hierdoor leren we namelijk zeer exact de werking van de microcomputer kennen. Voor de programmering van in- en outputroutines is het bijna een noodzaak om op machineniveau te programmeren. We hebben hierbij namelijk nogal eens te maken met tijdkritische programma's. We moeten soms snel op de acties van een extern apparaat reageren. Deze tijdkritische programma's zijn alleen maar te schrijven met behulp van machinecode, omdat deze wat betreft de executietijd vast liggen. Uiteraard verdient het daarbij de voorkeur via de assembler te werken, omdat dit de programmeur tijd spaart. Tevens kan zo commentaar in een programma worden verwerkt, zodat het programma leesbaarder wordt. Zie ook *Microprocessors 1.*

16.2. Machinestructuur

We moeten precies weten hoe de structuur van de machine er uit ziet, om een machine correct te kunnen programmeren. Dit wil zeggen, dat we exact moeten weten op welke adressen we random accessgeheugen hebben en op welke adressen we read only memory hebben. Ook moet bekend zijn op welke geheugen- (of I/O-) adressen de diverse poorten (of registers) van I/O-chips liggen. We moeten hiervoor een zogenaamde memory- en I/O-map maken. Als we deze adressen namelijk niet kennen, kunnen we nooit een datatransport naar (of vanuit) deze poorten realiseren en zo de machine besturen zodat deze werkt zoals wij dat willen. Mogelijke poorten in een personal computer zijn:

- Parallelle output-poorten, voor directe besturing.
- Parallelle input-poorten, voor directe besturing.
- Poorten met besturingsbits voor in- en output-interfaces.
- Poorten met besturingsbits voor motorbesturing van cassetterecorders.
- Het besturingsregister voor de USART, voor serie transmissie.
- Het statusregister van de USART, voor serie transmissie.
- Poorten van programmeerbare tellers, voor tijdtelling.
- Poorten voor de zogenaamde game paddle (besturingsknuppels), meestal ten behoeve van spelletjes.
- Poorten waarop de registers van een zogenaamde wetenschap-

pelijke reken-chip zijn aangesloten. Daarmee kan sneller gerekend worden.
– Poorten voor het aansturen en uitlezen van een keyboard.
– Poorten waarop de datatransmissiesnelheid voor een seriekanaal kan worden ingesteld.

De bovengenoemde poorten kunnen zowel als de I/O-mapped-I/O als Memory-Mapped-I/O zijn uitgevoerd. Bij de aansturing moet hiermee wel rekening worden gehouden. Het zal duidelijk zijn, dat al deze adressen en eventueel de bijbehorende codes van machine tot machine verschillen. Het is noodzakelijk om de bij een bepaalde machine horende documentatie nauwkeurig door te lezen en bij de hand te houden bij het schrijven van programma's voor besturing van de machine op machinetaal niveau. Het overzicht in afbeelding 16.1 laat de adressen zien, die gebruikt worden in de Teleac/DAI personal computer. Voor de diverse bits, die van belang zijn, wordt de betekenis aangegeven. Hierdoor wordt het mogelijk om straks in programma's te verwijzen naar de bits, die we op een bepaald ogenblik nodig hebben. De standaard Teleac/DAI personal computer heeft memory mapped-I/O. In hoofdstuk 17 gaan we daar nog uitgebreid op in.

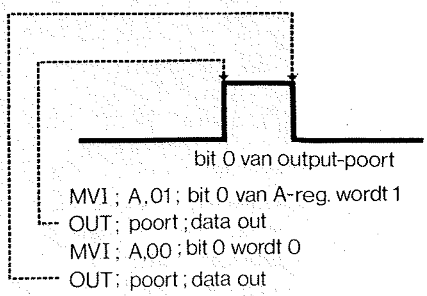
Functie	*Bit/woordfunctie*	*Adres*
REKENCHIP	**Commando/status**	**FB00/FB01**
AMD 9511	**Data**	**FB02/FB03**
GELUIDSGENERATOR	**Kanaal 0**	**FC00/FC01**
Intel 8253	**Kanaal 1**	**FC02/FC03**
	Kanaal 2	**FC04/FC05**
	Commando/status	**FC06/FC07**
BESTURING	–	**FD00–b0**
	–	**–b1**
	Page signal	**–b2**
	Serial out ready	**–b3**
	Right paddle	**–b4**
	Left paddle	**–b5**
	Random data	**–b6**
	Cassette input	**–b7**
BESTURING	**Paddle trigger**	**FD01**
BESTURING	**Volume geluid 0**	**FD04–b0/b3**
	Volume geluid 1	**–b4/b7**
	Volume geluid 2	**FD05–b0/b3**
	Volume random geluid	**–b4/b7**
BESTURING	**Cassette output**	**FD06–b0**
	Paddle selectie	**–b1/2**
	Paddle enable	**–b3**
	Cassette motor 1	**–b4**
	Cassette motor 2	**–b5**
	Rom bank switch	**–b6/b7**

PARALLEL I/O	**Port A**	**FE00**
Intel 8255	**Port B**	**FE01**
	Port C	**FE02**
	Commando/status	**FE03**
GEN.PURPOSE I/O	**Serie input buffer**	**FF00**
Texas Inst 5501	**Keyboard input**	**FF01–b0/b6**
	In7 DCE-bus	**–b7**
	Interrupt adr.reg.	**FF02**
	Status register	**FF03**
	Frame error	**–b1**
	Overrun error	**–b2**
	Receive buf. loaded	**–b3**
	transm. buf. empty	**–b4**
	Baud rate register	**FF05**
	Serie output buffer	**FF06**
	Keyboard output	**FF07**
	Interrupt mask reg.	**FF08**
	Timer 0	**FF09**
	Timer 1	**FF0A**
	Timer 2	**FF0B**
	Timer 3	**FF0C**
	Timer 4	**FF0D**

Afb. 16.1. Toewijzing van de poorten van de Teleac/DAI personal computer

16.3. Bitmanipulaties

Voordat we verder op het gebruik van het monitorprogramma ingaan, moeten we ons eerst nog bezinnen wat het betekent een eventueel bit van een geheugenplaats op 0 of 1 te zetten. Dit komt vaak voor bij het besturen van I/O-apparatuur. Wanneer we bijvoorbeeld een data-ready plus moeten geven aan een randapparaat kunnen we dit doen door achtereenvolgens een bit van een poort te setten ['1' maken) en te resetten ('0' maken). Afbeelding 16.2 geeft hiervan een voorbeeld.



Afb. 16.2. Genereren van een output-puls

We moeten voor I/O-besturing soms een programma schrijven, dat selectief een of meerdere bits hoog of laag kan maken zonder dat daarbij de andere bits van het woord veranderen. Hierbij kunnen we met succes gebruik maken van zogenaamde AND (EN) OR (OF) functies. Als we een bepaald bit op 0 willen zetten, moeten we de EN functie nemen van het woord, waarop we deze operatie willen uitvoeren, en een woord dat bestaat uit allemaal enen, behalve op de plaats waar we de betreffende bit 0 willen maken. Op deze plaats zal namelijk de code 0 moeten zijn (zie hiervoor afbeelding 16.3). Wanneer we een bepaalde bit van een woord hoog ('1') willen maken,

	b7	b6	b5	b4	b3	b2	b1	b0	
Oorspronkelijke data:	1	0	1	1	0	1	1	0	
OR masker (08 hexadecimaal):	0	0	0	0	1	0	0	0	OR
	1	0	1	1	1	1	1	0	
AND masker (8F hexadecimaal):	1	0	0	0	1	1	1	1	AND
Resultaat:	1	0	0	0	1	1	1	0	

Afb. 16.3. Toepassing van de AND en OR functies

betekent dit dat we van het betreffende woord de OF functie moeten nemen met een woord, dat alleen op die plaats waar we de 1 willen zetten een 1 heeft. De rest van de bits moet 0 zijn (zie hiervoor ook afbeelding 16.3). Als we deze techniek beheersen, is het mogelijk om een bepaalde geheugenplaats bit voor bit te setten of te resetten, net zoals we wensen. Deze technieken zullen vaak worden toegepast bij het programmeren van bepaalde I/O-apparatuur. In het bovenstaande voorbeeld wordt eerst bit 3 op 1 gezet en daarna de bits 4, 5 en 6 op 0 gezet. Dit doen we door het woord eerst te ORen met de hexadecimale code 08 en daarna het 8 bits woord te ANDen met de hexadecimale code 8F. De bitpatronen, waarmee we de AND en OR functies uitvoerden, worden *maskers* genoemd. Vaak zullen we dit soort methodieken in programma's gebruiken om een bepaald interface-protocol op parallelle basis te kunnen realiseren. AND en OR operaties kunnen zowel in machinetaal als in BASIC geprogrammeerd worden.

16.4. Opstarten van het monitorprogramma

Het opstarten van het monitorprogramma is voor iedere machine verschillend. Bij de Teleac/DAI personal computer kunnen we

vanuit de BASIC-werktoestand overgaan naar zogenaamde Utility Package door het commando UT (gevolgd door een carriage return) in te tikken. De computer zal hierop antwoorden door een zogenaamde prompt regel te geven. We nemen de Teleac/DAI computer als voorbeeld vanwege de uitgebreide monitor mogelijkheden. Een andere personal computer heeft meestal hier wat minder mogelijkheden. Na bestudering van de Teleac/DAI monitor is het dan gewoonlijk eenvoudig de monitor van een andere personal computer te begrijpen. In deze regel wordt verteld dat we ons in het Utility package bevinden. Hierin kunnen we Utility Package commando's intikken. Als we willen terugkeren vanuit het Utility Package naar de BASIC-interpreter, kunnen we dit doen door het B commando in te tikken. De computer komt dan direct terug met de melding BASIC, met daarachter een versie nummer. Dit versie nummer zegt niet met welke versie BASIC van de leverancier we te maken hebben. (In de SORCERER kunnen we vanuit BASIC overgegaan naar de monitor door BYE in te tikken. Vanuit de monitor kunnen we teruggaan naar BASIC door de letters PP (gevolgd door return) in te tikken. PP staat voor PROM PACKAGE.)

Monitorcommando's
Mogelijke commando's die we kunnen uitvoeren in de Teleac/DAI personal computer, zijn de volgende:

M : move, ofwel verplaats in het geheugen;
F : fill, ofwel vul geheugenplaatsen;
S : substitute, ofwel bekijk en vervang geheugeninhouden;
D : display of dump, ofwel laat geheugen inhouden zien;
G : go, ofwel starten van een programma;
L : look, ofwel bekijk de loop van het programma;
X : examine, ofwel bekijk en/of vervang registerinhouden;
W : cassette-write, ofwel schrijf naar cassette;
R : cassette-read, ofwel lees van cassette.

We gaan nu het overzicht, dat we zojuist hebben gegeven, nader uitwerken en van elk commando exact vertellen wat het voor effect heeft. We moeten hierbij echter nog de schrijfwijze van de commando's nader uitwerken. In de commando's komen namelijk aanduidingen voor adressen voor. De commando's die alle door een carriage return teken gevolgd moeten worden, zullen we daartoe de volgende toevoeging geven:

<CR>

De notatie voor de adressen is als volgt:

ADR : adres, een 16 bits adres;
LADR : laagste adres of startadres;
HADR : hoogste adres of stopadres;
DADR : bestemmingsadres of eindadres.

Voor het aangeven van data hebben we:

BYTE : 8 bits datawoord.

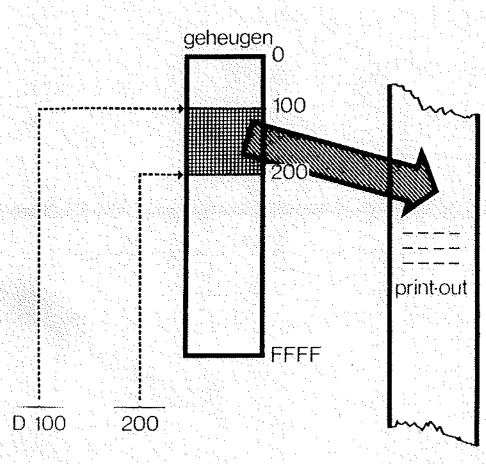
Alle adressen en byte-codes worden opgegeven door middel van hexadecimale getallen. We moeten voor adressen 4 hexadecimale cijfers opgeven en voor een byte 2 hexadecimale cijfers. Wanneer we om een of andere reden bij het opgeven van hexadecimale getallen een tikfout hebben gemaakt, kunnen we alsnog de juiste code intikken, want de machine zal alleen de laatste 4 (of 2) cijfers nemen, net wat er voor een bepaald commando van toepassing is. Dit geeft ons de mogelijkheid om snel fouten te corrigeren voordat we de carrier return geven. De argumenten van een commando mogen we scheiden door een spatie of een komma. De argumenten van een commando zijn de getalwaarden, die moeten worden ingetypt. We kunnen 3 groepen commando's onderscheiden namelijk:

- geheugen- en registercommando's;
- executiecommando's;
- cassettecommando's.

Geheugencommando's
DISPLAY (zichtbaar maken)

D LADR HADR <CR>

Door middel van het DISPLAY-commando wordt het mogelijk om de inhoud van een aaneengesloten deel van het geheugen op het scherm zichtbaar te maken. In de afbeeldingen 16.4 en 16.5 zien we een voorbeeld van het toepassen van het *display*-commando. Hierbij



Afb. 16.4. Display-commando

```
    UT
PC UTILITY V3.3
>D500 57F

0500 00 08 00 08 00 08 00 00 00 00 00 00 00 08 00 00
0510 00 08 00 08 00 68 00 00 00 68 00 60 88 68 08 E0
0520 00 08 00 88 02 88 02 88 01 C8 00 80 00 08 02 0A
0530 00 00 00 00 00 0A 02 28 08 6A 00 4A 00 49 41 80
0540 00 00 00 81 00 88 08 68 00 60 00 62 00 68 01 60
0550 20 00 01 40 00 08 02 4A 21 08 8B 48 21 00 28 08
0560 20 00 89 60 21 28 08 08 00 00 00 00 20 28 08 68
0570 02 38 00 48 00 00 00 00 00 00 00 08 00 00 08 00
>
```

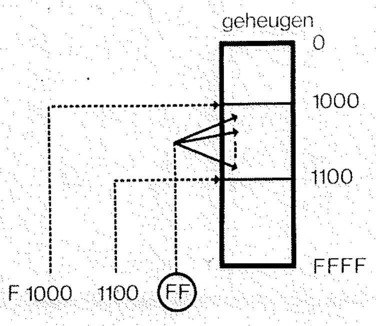
Afb. 16.5. Display

wordt de geheugeninhoud van de plaatsen 500 tot en met 57F zichtbaar gemaakt. Zo kunnen we testen of de geheugeninhoud inderdaad is zoals we verwachten.

FILL (vullen)

F LADR HADR BYTE <CR>

Met behulp van het FILL-commando kunnen we een bepaald aaneengesloten geheugenblok vullen met een opgegeven waarde (BYTE). Hierbij wordt elke geheugenplaats vanaf het beginadres tot en met het eindadres gevuld met deze waarde. De afbeeldingen 16.6 en 16.7 geven een voorbeeld van het toepassen van het FILL-commando. De geheugenplaatsen 500 tot en met 57F (hexadecimaal) vullen we met nullen. Daarna worden de geheugenplaatsen 550 tot en met 56F gevuld met enen. De inhoud van het gehele geheugenblok wordt zichtbaar gemaakt met behulp van het DISPLAY-commando zowel voor als na de uitvoering van FILL. Het is bij het testen vaak nuttig met een bekende geheugeninhoud te starten. Er kan dan nagegaan worden waar het programma precies schrijft en wat er geschreven wordt.



Afb. 16.6. Fill-commando

```
D500 57F

0500 00 08 00 08 00 08 00 00 00 00 00 00 00 08 00 00
0510 00 08 00 08 00 68 00 00 00 68 00 60 88 68 08 E0
0520 00 08 00 88 02 88 02 88 01 C8 00 80 00 08 02 0A
0530 00 00 00 00 00 0A 02 28 08 6A 00 4A 00 49 41 80
0540 00 00 00 81 00 88 08 68 00 60 00 62 00 68 01 60
0550 20 00 01 40 00 08 02 4A 21 08 8B 48 21 00 28 08
0560 20 00 89 60 21 28 08 08 00 00 00 00 20 28 08 68
0570 02 38 00 48 00 00 00 00 00 00 00 08 00 00 08 00
>F500 57F 00

>F550 56F FF

>D500 57F

0500 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0510 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0520 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0530 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0540 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0550 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
0560 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
0570 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
>
```

Afb. 16.7. Fill

SUBSTITUTE (substitutie)

S ADR <CR>

Met behulp van het SUBSTITUTE-commando kunnen we een reeks geheugenplaatsen vanaf het opgegeven adres veranderen. Als we het

bewuste adres hebben opgegeven en de returntoets hebben ingedrukt, zal de computer de inhoud van het betreffende adres zichtbaar maken. We kunnen daarna een nieuwe hexadecimale waarde intikken (hiervan worden weer slechts de laatste 2 hexadecimale cijfers overgenomen) en dit laten we volgen door een spatie. Zo kunnen we geheugenplaats na geheugenplaats veranderen of eventueel gelijk laten. Als we niet willen veranderen tikken we direct een spatie in. Het monitorprogramma zal dan naar de volgende plaats gaan. Als we het S commando willen afbreken, kunnen we dit doen door een niet toegelaten hexadecimale waarde in te tikken, bijvoorbeeld een s of deelstreep of iets dergelijks. In afbeelding 16.8 zien we een toepassing van het SUBSTITUTE-commando. Hierbij veranderen we plaats 520 tot en met 52F van het geheugen met behulp van het SUBSTITUTE-commando. Via een DISPLAY zien we wat er is gebeurd. Zo kunnen we bijvoorbeeld een teken een gewenste beginwaarde geven, opdat we sneller een te testen effect willen bereiken.

```
D500 57F

0500 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0510 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0520 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0530 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0540 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0550 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
0560 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
0570 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
>S520
00-12 00-13 00-23 00-24 00-34 00-45 00-46 00-56 00-11
00-22 00-22 00-44 00-AA 00-FF 00-EE 00-88 00-/
?
>D500 57F

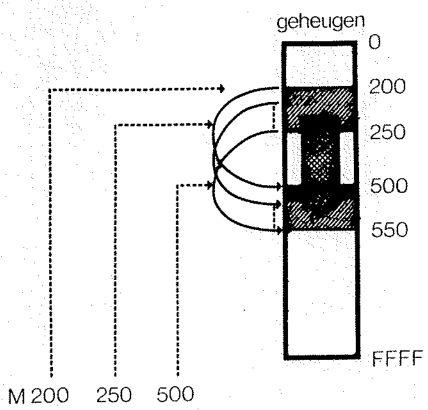
0500 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0510 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0520 12 13 23 24 34 45 46 56 11 22 22 44 AA FF EE 88
0530 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0540 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0550 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
0560 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
0570 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

Afb. 16.8. Substitute

MOVE (verplaatsen)

M LADR HADR DADR <CR>

Met behulp van het MOVE-commando kunnen we een aaneengesloten geheugenblok, dat loopt van het lage adres (LADR) naar het hoge adres (HADR), verplaatsen naar een ander geheugenblok, dat begint bij het destinationadres (DADR). Afbeeldingen 16.9 en 16.10 verklaren een en ander nader. Hierbij wordt de inhoud vanaf geheu-



Afb. 16.9. Move-commando

```
D500 57F

0500 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0510 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0520 12 13 23 24 34 45 46 56 11 22 22 44 AA FF EE 88
0530 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0540 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0550 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
0560 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
0570 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
>M520 52F 560

>D500 57F

0500 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0510 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0520 12 13 23 24 34 45 46 56 11 22 22 44 AA FF EE 88
0530 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0540 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0550 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
0560 12 13 23 24 34 45 46 56 11 22 22 44 AA FF EE 88
0570 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
>
```

Afb. 16.10. Move

genplaats 520 tot en met 52F verplaatst naar plaats 560 tot en met 56F. Zo kunnen we later deze inhoud met een nieuw verkregen inhoud vergelijken.

EXAMINE (bekijken van registers)

> X <CR>
> of:
> XR<CR>
> met R een van de registers (A, B, C, D, E, F, H, L), programmateller (P), stackpointer (S) of een van de speciale machineregisters zoals het interrupt maskerregister (M), timercounter commandowoord (T) en het parallel I/O-commandowoord (G). De M, T en S van de Teleac/DAI computer stellen we niet verder aan de orde. Voor bezitters van deze computer verwijzen we naar de gebruikershandleidingen.

Als we het X-commando geven, zal de machine daarop direct de inhoud van alle registers, de programmateller, de stackpointer en de waarde van de speciale registers laten zien. In afbeelding 16.11 zien we hiervan een voorbeeld. Het is ook mogelijk om slechts een onderdeel van de registers te bekijken door de X te laten volgen door de letter, die aangeeft welk register we willen hebben. Zo zal XD laten zien wat de inhoud van het D register is en we kunnen nu (net als bij het SUBSTITUTE-commando ingeval van geheugen) de waarde van een register veranderen door daarvoor een nieuwe waarde in te tikken. Als we een spatie intikken zal daarna het volgende register, dus in ons geval het register E, aan de orde komen. Deze reeks kunnen

we weer afbreken door een niet toegelaten hexadecimale code in te tikken. Dit geeft ons de mogelijkheid om zo alle registers te veranderen in een waarde, die we voor een programma getest willen hebben. Omdat vele instructies met registers werken is dit commando bijzonder waardevol. Steeds kan bezien worden of de registers die waarden bevatten, die we bij het programmeren voor ogen hadden.

```
 X
A=00 F=46 B=01 C=40 D=80 E=8F H=E6 L=9E S=F8F8 P=0000
>XA 00-11
46-22
01-33
40-44
80-55
8F-66
E6-77
9E-88
F8F8-F8F8
0000-500

>X
A=11 F=22 B=33 C=44 D=55 E=66 H=77 L=88 S=F8F8 P=0500
>
```

Afb. 16.11. Examine

Executiecommando's

GO (starten)

G ADR <CR>

Als we een GO-commando geven met daarbij een opgegeven adres, zal de machine het programma (in machinecode) dat begint op het aangegeven adres ten uitvoer brengen. De machine zal nu niet meer op besturing van de monitor staan maar geheel aan de hand van het aangegeven programma werken. Het voert daarbij een gebruikersprogramma uit. Als we geen adres achter het G-commando geven, zal de machine automatisch het adres nemen dat in de programma-teller aanwezig is. Die waarde kunnen we ook zetten met XP (zie onder het X-commando). Een voorbeeld van het G-commando zien we hieronder.

G 1000

LOOK (programma-onderzoek)

L ADR LADR HADR <CR>

Met behulp van het LOOK-commando is het mogelijk om een programma, beginnend bij een aangegeven adres (ADR), op te starten. Tijdens de executie wordt getoond wat er zoal met de registers gebeurt. In het LOOK-commando wordt namelijk door LADR en HADR aangegeven in welk stuk van de executie we geïnteresseerd zijn. De computer laat dan stap voor stap zien wat er zoal met de registers gebeurt. LADR is het laagste adres waarop dit gebeurt, HADR het hoogste. We laten dus zien wat er gebeurt als we instructies uitvoeren tussen LADR en HADR. In het het voorbeeld wordt

```
*UT
PC UTILITY V3.3
>S1000
FF-3E
7F-00
FF-3C
FF-C3
FF-02
7F-10
FF-/
?
>Z3
```

Afb. 16.12. Look (Look werkt alleen op video-scherm, niet op een printer-scherm)

eerst de inhoud van de accumulator 0 gemaakt en daarna in een lus telkens met 1 verhoogd. We gaan dit verhogen nu bekijken. Hiervoor stellen we het L-commando in als aangegeven in afbeelding 16.12. Het programma wordt opgestart op adres 1000 en stap voor stap zien we wat er gebeurt. Omdat dit een programma is dat nooit zal eindigen, stoppen we het programma met een BREAK. Hiervoor drukken we de BREAK toets in. Voor dat we het L-commando kunnen uitvoeren zullen we eerst de machine goed moeten initialiseren. D.w.z. in een door ons bepaalde toestand brengen. De machine wordt namelijk telkens gestopt nadat een instructie is uitgevoerd.

Tapecommando's
WRITE (schrijven)

W LADR HADR <CR>

READ (lezen)

R <CR>

Om machinetaal programma's in hexadecimale vorm voor later gebruik te kunnen bewaren, kunnen we ze net als een BASIC-programma vanuit de machine op een cassetteband opslaan. Hiervoor moeten we aan de machine het begin- en het eindadres van het geheugenstuk, dat we willen opslaan, aangeven. Het W-commando zal het wegschrijven verzorgen. Later kunnen we dan door middel van het R-commando de opgeslagen data weer vanaf de band in het geheugen inlezen. Het W-commando moeten we laten volgen door het begin- en eindadres. Het R-commando heeft geen adresaanduiding nodig. Het leest namelijk begin- en eindadres weer van de band. In afbeelding 16.13 zien we achtereenvolgens het zichtbaar maken van de inhoud van een geheugenblok, het wegschrijven van dat blok, het overschrijven van dit geheugenblok door middel van een FILL-commando, daarna het lezen van de tape en het telkens zichtbaar maken van de inhoud van dat geheugenblok.

```
D1000 100F

1000 3E 00 3C C3 02 10 FF FF FF 7F FF 7F FF 7F BF 7F
>W1000 1005

SET RECORD,START TAPE,TYPE SPACE

>F1000 100F 00

>D1000 100F

1000 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
>R

>D1000 100F

1000 3E 00 3C C3 02 10 00 00 00 00 00 00 00 00 00 00
>
```

Afb. 16.13. Tapecommando

(De EXIDY SORCERER heeft in plaats van een D-commando de letters DU, gevolgd door twee hexadecimale adressen. Voor het SUBSTITUTE-commando vinden we daar ENTER-commando (EN). Voor het write naar tape vinden we save SA, voor het teruglezen read hebben we LO van load. Voor go moeten we de letters GO intikken in plaats van alleen G, voor move moeten we MO intikken, gevolgd door drie adressen. Verder komen de commando's wat betekenis betreft vrij goed overeen.)

16.5. Ingebouwde subroutines

Gewoonlijk stelt een personal computer ook ingebouwde subroutines (I/O drivers) ter beschikking van de programmeur om de al aanwezige standaard randapparatuur te bedienen. Voorbeelden zijn drivers voor het toetsenbord, de cassette-recorder, een printer etc. Raadpleeg voor deze mogelijkheden uw handleiding. We hebben ons hier beperkt tot monitor routines die vanaf het toetsenbord kunnen worden opgeroepen. Soms vat men de ingebouwde subroutines ook op als deel van de monitor.

Samenvatting hoofdstuk 16

In dit hoofdstuk is een aantal hulpmiddelen behandeld, die van groot nut kunnen zijn bij het maken en testen van programma's. Met behulp van de besproken technieken kan tot diep in de machine worden ingegrepen. Het is zelfs mogelijk zo de normale werking te veranderen. Er kan direct in het geheugen worden ingegrepen. We zouden dus ook bijvoorbeeld het videogeheugen of de I/O-software kunnen veranderen.

Multiple choice-vragen bij hoofdstuk 16

A. Wat is fout?
Een monitorprogramma dient om:

1 fouten te zoeken
2 geheugeninhoud te modificeren
3 registers te lezen
4 direct input- en outputinstructies uit te voeren

B. **a.** De machine-structuur is niet van belang voor de assemblerprogrammeur.
b. Alle microcomputers hebben een gelijke machine-opbouw.

1 uitspraak a is goed en uitspraak b is goed
2 uitspraak a is fout en uitspraak b is goed
3 uitspraak a is goed en uitspraak b is fout
4 uitspraak a is fout en uitspraak b is fout

C. Bij memory-mapped output-poorten kunnen we met behulp van het monitorprogramma de output-data in de poorten schrijven met het:
a. substitute commando
b. move commando
c. display commando
Correcte uitspraken zijn:

1 a en b
2 b en c
3 a en c
4 a, b en c

D. We kunnen met behulp van het monitorprogramma register A een initiële waarde 0B (hex.) geven door middel van het:

1 substitute commando
2 move commando
3 examine commando
4 fill commando

E. Registers nul maken gebeurt met het:

1 substitute commando
2 move commando
3 examen commando
4 fill commando

F. **a.** Programma-ontwikkeling met behulp van een monitorprogramma is niet aan te raden!
b. Met een monitor-commando kunnen we een symbolische programma-listing maken.

1 uitspraak a is goed en uitspraak b is goed
2 uitspraak a is fout en uitspraak b is goed
3 uitspraak a is goed en uitspraak b is fout
4 uitspraak a is fout en uitspraak b is fout

G. **a.** We kunnen met behulp van het monitorprogramma een programma, dat in het RAM-geheugen van de computer staat, verbeteren.
b. Een monitorprogramma berekent zelf de adressen van labels in programma's.

1 uitspraak a is goed en uitspraak b is goed
2 uitspraak a is fout en uitspraak b is goed
3 uitspraak a is goed en uitspraak b is fout
4 uitspraak a is fout en uitspraak b is fout

H. In het programma van afb. 16.2 kunnen we de duur van de plus regelen door:

1 tussen de tweede en de derde instructie een HLT te zetten
2 tussen de tweede en de derde instructie een wachtlus te zetten
3 in plaats van MVI A,01H te zetten MVI A, 08H
4 in plaats van MVI A,00H te zetten MVI A,0FFH

I. Met behulp van de monitor-functie LOOK kunnen we:

1 geheugeninhouden bekijken
2 programma-stappen bekijken
3 I/O-data bekijken
4 stackinhoud bekijken

J. We willen de geheugenplaats 1000 (hex) en volgende vullen met de code voor MVI A,0FFH
Dit doen we met het substitute commando. Welke code moeten we voor de instructie invoeren?

1 7F FF
2 3A FF
3 3E FF
4 MA FF

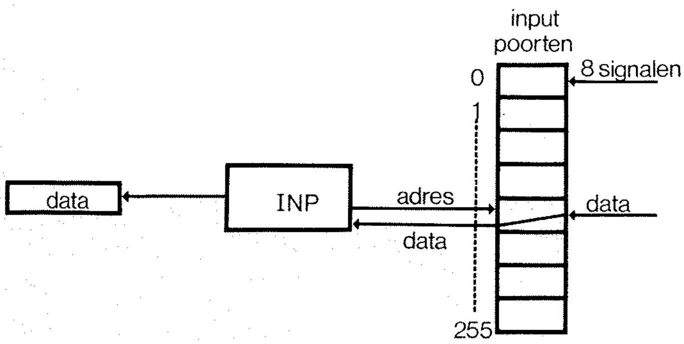
Hoofdstuk 17

I/O-poorten in BASIC

In dit hoofdstuk bespreken we hoe de invoer en uitvoer (I/O) via I/O-poorten in BASIC wordt behandeld. Daarbij gaan we in op hanteerbare concepten. Daarnaast worden de voor dit doel specifieke BASIC instructies besproken. Tenslotte wordt aangegeven welke beperkingen we ondervinden en hoe die eventueel via programma's in de machinetaal te omzeilen zijn.

17.1. INP en OUT

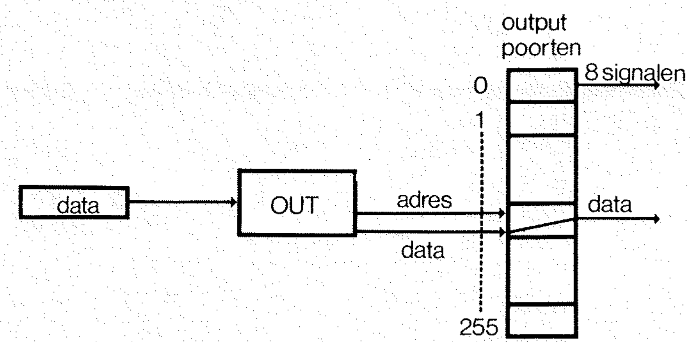
In de hoofdstukken 14 en 15 hebben we in principe gezien hoe I/O op parallelle wijze of in serie kan worden gehanteerd. Daarbij is ook het begrip *input-poort* en *output-poort* geïntroduceerd. Via een input-poort kunnen we een 8 bits binaire waarde van de buitenwereld (dat wil zeggen: van buiten het microcomputersysteem) naar binnen halen. We gaan nu eerst uit van I/O-mapped-I/O. We moeten dan in de Teleac/DAI personal computer 1 uit de 256 mogelijke input-poorten adresseren. Via een output-poort kunnen we een 8 bits waarde naar buiten voeren. Ook dan moeten we weer 1 uit de 256 mogelijke output-poorten adresseren. De waarde, die we naar een output-poort hebben gestuurd, blijft daar staan totdat we een nieuwe waarde naar dezelfde poort sturen. Wanneer we in machinetaal programmeren, weten we dat de plaats, waar deze waarde naar toe gaat, respectievelijk vandaan komt, in de 8080 microprocessor de accumulator is. Deze wordt ook wel als register A aangeduid.
In BASIC kennen we echter de accumulator en de overige registers niet. Het begrip 'adressering' komt in die vorm niet in BASIC voor. Vooral daarom hebben vele BASIC versies op microcomputers toch naar een gelijkwaardige oplossing gezocht. Daarbij herkennen we weer het begrip 'functie' en 'input en output'.
We creëren een input-functie INP(I), die als argument het adres van de input-poort meekrijgt. Zoals bij iedere functie kunnen we het resultaat ervan weer in expressies gebruiken. (Zie afbeelding 17.1). Op deze manier kunnen we, evenals bij assembler programmering de input van de buitenwereld rechtstreeks verwerken. We moeten wel opmerken, dat hierbij de BASIC interpreter eerst het statement moet herkennen, dan de functie verrichten en tenslotte het resultaat ergens aan toekennen. De snelheid waarmee waarden achter elkaar kunnen worden binnengehaald, is daardoor aanzienlijk lager dan bij machinetaalprogrammering. Daarin gebruiken we namelijk de instructie IN 1, die na vertaling door de assembler in machinecode in de 8080 microcomputer is gezet. Voor de duidelijkheid is hier-




```
*LIST
1Ø    LET ADRES=1.Ø
2Ø    LET A=INP(ADRES)
3Ø    PRINT A
*RUN
 Ø.Ø
```

Afb. 17.1. De INP functie

boven steeds een apart statement gebruikt. In plaats van regels 1Ø, 2Ø en 3Ø zou ook 1Ø PRINT INP(1) hetzelfde resultaat opleveren.
BASIC kan niet rechtstreeks op binaire getallen opereren. Het is dus ook niet verwonderlijk dat de waarde, die door de functie INP(I) wordt afgeleverd, een decimale en geen binaire waarde is. Deze geeft de decimale waarde van het opgehaalde byte weer, waarbij ervan uitgegaan is dat deze waarde 8 bits binair gecodeerd is.
Wanneer de opgehaalde 8 bits ØØØØØØ1Ø zijn, zal als resultaat van de functie de integer 2 ter beschikking komen.
Vanzelfsprekend bestaat er ook voor de uitvoer een vergelijkbare functie. Dit is de OUT I, J functie.
Deze werkt op 2 argumenten. De eerste geeft het adres van de output-poort weer, terwijl de tweede de waarde weerspiegelt die moet worden uitgevoerd (zie afbeelding 17.2).




```
*LIST
1Ø    LET ADRES=1.Ø
2Ø    LET WAARDE=9.Ø
3Ø    OUT ADRES,WAARDE
*RUN
*
```

Afb. 17.2. De OUT functie

Het bitpatroon, dat in dit voorbeeld naar output-poort 1 wordt gestuurd, is ØØØØ1ØØ1 (de binaire representatie van 9).
Hier moeten we de parameters weer in decimale vorm aanbieden. Ook nu weer zal deze bewerking via de interpreter trager verlopen dan via de vergelijkbare assembler instructie OUT 1, waarbij de uit te voeren waarde in binaire vorm in de accumulator aanwezig is. Dat de uitvoer en invoer van veel toepassingen op deze wijze wat traag verlopen, is geen duidelijk nadeel, als de buitenwereld op zich ook traag is. Dit geldt met name als op mechanische of menselijke handelingen vanuit die buitenwereld moet worden gereageerd. In tijdkritische situaties is het gebruik van INP(I) en OUT I, J echter niet aan te raden.

17.2. Voorbeeld, een looplicht

We zullen nu een eenvoudig voorbeeld behandelen, waarbij we 8 lampen achtereenvolgens aan en uit laten gaan om dan weer van voren af aan te beginnen. Dit wordt wel eens een *looplicht* genoemd.

```
*LIST
10    REM DEMONSTRATIE PROGRAMMA VAN EEN LOOPLICHT
20    REM OUTPUTPOORT 1
30    LET ADRES=1.0
40    FOR I=0.0 TO 7.0
50    LET WAARDE=1.0 SHL I
60    OUT ADRES,WAARDE
70    NEXT I
80    GOTO 40
*RUN
```

Aan het programma is te zien dat dit eindeloos door blijft lopen. Alleen via de BREAK toets kunnen we bij de Teleac/DAI personal computer het programma onderbreken.
Opmerkelijk is hier het gebruik van de operator SHL (SHIFT LEFT). Voor X SHL Y geldt dat de binaire waarde van X over Y plaatsen naar links wordt geschoven. Bij ons voorbeeld is X = 1 en Y respectievelijk Ø tot en met 7, zodat de waarden 1, 2, 4, 8, 16, 32, 64, 128 ontstaan. We hadden hier ook wel WAARDE = 2↑I kunnen gebruiken. Dit geeft hetzelfde resultaat.
We laten nu ook zien hoe hetzelfde programma er in assemblertaal geschreven uitziet. Daarvoor geven we de listing van de assembler weer:

```
0001              ADRES    EQU   1      ;ADRES VAN DE OUTPUTPOORT
0000                       ORG   1200H  ;BEGINADRES VAN HET PROGRAMMA
1200   3E01       BEGIN:   MVI   A,01H  ;ZET 1 IN DE ACCU
1202   D301       WEER:    OUT   ADRES  ;STUUR ACCUWAARDE NAAR DE OUTPUTPOORT
1204   07                  RLC          ;VERSCHUIF ACCUWAARDE 1 PLAATS NAAR
                                        ;LINKS.
1205   C30212              JMP   WEER   ;OPNIEUW SCHUIVEN EN UITVOEREN
1200                       END   BEGIN

NO PROGRAM ERROR(S)
```

Het is daarbij niet mogelijk het programma met de BREAK toets te onderbreken. Het kijken naar de Break toets gebeurt in de interpreter. Als we eigen machinetaalprogramma's runnen is deze interpreter niet actief. Om enigszins te kunnen volgen wat er gebeurt, laten we op regel 65 de uitgevoerde waarde via een TAB-functie printen, gevolgd door! Nadat we de 8 lampen eenmaal aangezet hebben, beginnen we opnieuw via regel 8Ø. De waarden die we op regel 5Ø produceren moeten uiteraard 1 2 4 8 enz. zijn.

```
*LIST
10      REM DEMONSTRATIE PROGRAMMA VAN EEN LOOPLICHT
20      REM OUTPUTPOORT 1
30      LET ADRES=1.0
40      FOR I=0.0 TO 7.0
50      LET WAARDE=1.0 SHL I
60      OUT ADRES,WAARDE
65      PRINT TAB(60-I*8);"!"
70      NEXT I
80      GOTO 40
*RUN
                                                        !
                                                !
                                        !
                                !
                        !
                !
        !
!
                                                        !
                                                !
                                        !
                                !
```

Wanneer we de lamp van links naar rechts willen laten lopen, moeten we regel 5Ø vervangen door:

5Ø LET WAARDE = 128 SHR I (shift right) of ook wel:
5Ø LET WAARDE = 2 ↑ (7 − I)

Als we het lopen langzamer willen laten gebeuren, voegen we tussen regels 6Ø en 7Ø nog een zgn. dummy loop toe. Deze doet in feite niets anders dan vertragen. We voeren de berekening alleen maar uit om tijd te verliezen.

Voorbeeld

```
*LIST 61-65
61      REM VERTRAGINGSLOOP
62      LET V=0.24
63      FOR W=1.0 TO 250.0
64      LET A=V*V/V
65      NEXT W
*
```

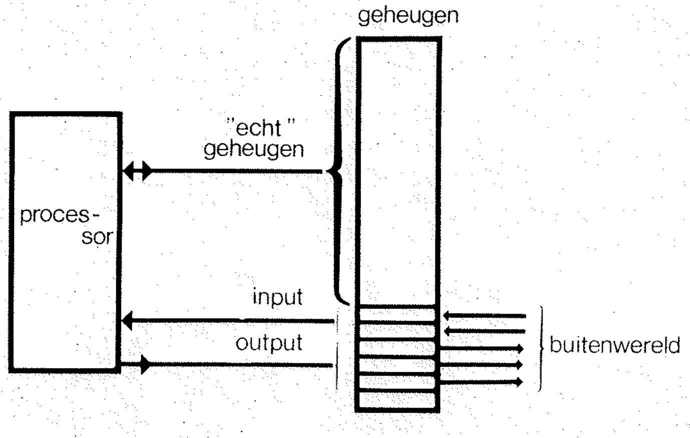
Met dit soort vertragingsloops kunnen we de lampen zo langzaam laten lopen als we willen.
Bij machinetaal programmering is van iedere instructie precies bekend hoeveel tijd de uitvoering ervan kost.
Bij gebruik van BASIC is dat niet bekend. Het is sterk afhankelijk van de interpreter, die bij het systeem is gekozen, hoe snel operaties worden uitgevoerd. Aangezien de gebruiker niet weet hoe de interpreter exact werkt, is hij in het algemeen niet in staat aan exacte timingseisen te voldoen. Uiteraard kan wel bij een eenmaal gekocht systeem, dat niet meer gewijzigd wordt, gemeten worden hoe lang

een bepaald programma (bijvoorbeeld het bovenstaande programma) over de uitvoering doet. Het is veel moeilijker dit per operatie exact te meten. Hier hebben we veel meer het *trial-and-error systeem* nodig, dat gebaseerd is op 'probeer het maar net zo lang totdat het goed is'.
We hebben dus voor I/O in BASIC 2 functies ter beschikking:

INP (I) haal de waarde uit input-poort I op
OUT I, J brengt de waarde J naar output-poort I

De waarde I en J moeten daarbij in het bereik van Ø tot en met 255 liggen. Verder moeten we precies weten welke adressen van de input- en output-poorten beschikbaar zijn voor de BASIC programmeur. Het systeem zal namelijk ook zelf I/O-poorten gebruiken voor de bediening van het toetsenbord, het display, de cassetterecorder, enz. Sommige poorten zullen zelfs niet echt in het systeem in hardware aanwezig zijn. Zie hiervoor ook hoofdstuk 16.
In de hoofdstukken 13, 14 en 15 hebben we gezien hoe I/O-poorten in principe in een systeem aangesloten zijn aan de adres-, data- en controlbus.
Voor een input-poort zijn daarvoor een adresselectie-schakeling, het controlesignaal IN en de verbinding met de databus nodig.
Voor een output-poort wordt in plaats van het controlesignaal IN het signaal OUT gebruikt. In de hoofdstukken 14 en 15 is uitgebreid ingegaan op de werkelijke componenten voor parallelle I/O.
De methode om I/O via INP en OUT statements te verwerken wordt wel I/O mapped I/O genoemd.
Bij I/O mapped I/O moet de programmeur gebruik maken van de INP en OUT instructies. Om na te gaan welke adressen hij daarbij voor I/O-poorten kan gebruiken, moet hij de handleiding van zijn computersysteem raadplegen waarin die adressen staan aangegeven. Gewoonlijk zal slechts een deel van de mogelijke 256 adressen in het systeem zijn gebruikt. De handleiding moet ook aangeven op welke aansluitpen van de aansluitstekkers (gewoonlijk *connector* genoemd) ieder bit van de aanwezige I/O-poorten is aangesloten.
Het is in het programma volstrekt duidelijk dat er I/O behandeld wordt. De instructies INP en OUT worden dan gebruikt. Geheugen en I/O worden als afzonderlijk beschouwd. De maximale hoeveelheid geheugen en de maximale hoeveelheid I/O-poorten kan in principe in de computer worden benut.
Wanneer we bij de 8080 microcomputer voor de adresselectie alle 16 bits van de adresbus gebruiken en in plaats van de IN en OUT signalen de controlesignalen READ en WRITE, kunnen we de I/O-poorten als onderdeel van het geheugen beschouwen. We hebben dan te maken met de methode van memory-mapped-I/O, zoals in vorige hoofdstukken is uitgelegd.
Bij memory-mapped-I/O ligt voor de programmeur de situatie anders. De I/O-poorten worden als deel van het geheugen beschouwd. Er zijn geen afwijkende instructies voor I/O. De instructies die op het geheugen betrekking hebben, hebben ook betrekking op de I/O-poorten (zie afbeelding 17.3).
Ook nu moet de programmeur de handleiding van zijn computer raadplegen. Hij moet daarin nagaan welke geheugenadressen betrekking hebben op echt geheugen en welke adressen in feite I/O-poorten vertegenwoordigen. Deze twee soorten moet hij dan in zijn

programma zorgvuldig uit elkaar houden. In het programma zelf is niet zonder meer te herkennen of er I/O wordt gebruikt.
Beide hier genoemde methodes voor I/O worden wel toegepast, zowel via assembler programmering als via BASIC. De Teleac/DAI personal computer kan in feite van beide methodes gebruikmaken. Voor de I/O-aansluitingen, die standaard in de Teleac/DAI personal computer aanwezig zijn, wordt memory-mapped-I/O gebruikt.
Voor uitbreiding van het systeem met de zogenaamde DAI Real World Cards wordt I/O mapped I/O gebruikt. Deze kaarten geven een veelheid van I/O mogelijkheden.



Afb. 17.3. Memory-mapped I/O

De I/O mapped I/O-methode is hier eerst behandeld, omdat deze het eenvoudigst weergeeft wat er precies moet worden gedaan, hoewel de standaard Teleac/DAI methode zonder uitbreidingen niet gebruikt kan worden.
We gaan nu verder in op het gebruik van geheugenadressen in BASIC. Hier doet zich het probleem voor dat BASIC normaal niet met geheugenadressen werkt. Het begrip geheugenadres is niet bekend, noch het begrip I/O-adres. Vandaar dat in veel personal computers een uitbreiding voor BASIC heeft plaatsgevonden om dat probleem op te lossen.

17.3. PEEK

Wanneer we informatie van uit een bepaalde geheugenplaats willen lezen gebruiken we de functie PEEK (I).

Voorbeeld

```
*LIST
10    LET ADRES=3.0
20    PRINT PEEK(ADRES)
*RUN
 98
*
```

We zien hier weer dat het argument waarop PEEK opereert, een adres is. Dit adres verwijst naar een plaats in het geheugen van de computer.
In bovenstaand voorbeeld lezen we de informatie, die staat op de geheugenplaats, welke het adres 3 heeft.

Het resultaat is evenals bij het INP statement een decimale waarde, dat wil zeggen het geeft de binaire codering van die bepaalde geheugeninhoud decimaal weer. De algemene vorm van PEEK is:

PEEK (I)

waarbij I het bereik heeft van Ø tot en met 65 535 (16 bits).
De 8080 microprocessor werkt namelijk met een 16 bits adres, hetgeen 65 536 mogelijkheden biedt om een geheugenplaats te adresseren. De decimale waarde van het resultaat ligt uiteraard weer in het bereik van Ø tot en met 255, want een geheugenwoord is 8 bits breed. Met behulp van de PEEK statements is het mogelijk om de in machinecode gestelde instructie uit het geheugen te lezen. Ook kunnen gegevens uit gespecificeerde geheugenplaatsen gelezen worden, enz. Het is daarbij natuurlijk wel zaak om precies te weten waar zich de gewenste geheugenplaatsen in de computer bevinden. In het geval van memory-mapped-I/O moet daarbij ook in de gaten worden gehouden, welke geheugenplaatsen in feite I/O-plaatsen zijn. Die laatsten hebben namelijk direct contact met de buitenwereld.

17.4. POKE

Met PEEK kunnen we uit geheugenplaatsen informatie verkrijgen. Met de functie POKE schrijven we informatie naar de gespecificeerde geheugenplaatsen. Daarvoor moeten we niet alleen de te schrijven waarde, maar ook het geheugenadres als argument opgeven.

Voorbeeld

```
*LIST
1Ø     LET WAARDE=15.Ø
2Ø     LET ADRES=Ø.Ø
3Ø     POKE ADRES,WAARDE
*RUN
*PRINT PEEK(Ø)
 15
*
```

In dit voorbeeld wordt de binaire waarde ØØØØ1111 (dat decimaal 15 voorstelt) op geheugenplaats Ø neergeschreven.
De waarde, die moet worden geschreven, ligt tussen Ø en 255 en het adresbereik is weer van Ø tot en met 65 535.
Voor het gebruik van het POKE statement is bijzondere zorgvuldigheid vereist. In de eerste plaats moet de programmeur goed weten wat hij doet. Hij moet weten waar in het systeem geheugen aanwezig is. Bij memory-mapped-I/O moet hij ook beseffen dat de I/O-plaatsen de informatie rechtstreeks naar de buitenwereld zenden. Wanneer dat per vergissing gebeurt, kan die buitenwereld daardoor volledig ontregeld raken.
Wanneer we informatie per vergissing naar een geheugenplaats zenden, die niet aanwezig is, kan dat geen kwaad, hoewel we de informatie wel kwijt zijn. Wanneer we later door middel van PEEK de informatie proberen terug te lezen, zal dat niet lukken.
Een andere mogelijkheid is dat de BASIC interpreter volledig ontregeld raakt. Het systeem is dan op dat moment niet meer in staat goed te reageren. Eventuele in het geheugen aanwezige programma's gaan

verloren. Alleen door een herstart (wat soms inhoudt: de machine uitzetten en daarna weer aanzetten) kan de machine weer in het gareel worden gebracht.
Waarom kan dit dan wel voorkomen? De BASIC interpreter is in ROM aanwezig, zodat die niet kan worden vernield door foutieve instructies. Het BASIC programma zelf staat echter in RAM, evenals de waarden en de namen van de variabelen. Verder staan er allerlei voor het systeem benodigde verwijzingen in het RAM geheugen zoals beginpositie en lengte van een string, enz. (zie hoofdstuk 12). Het is duidelijk dat we in dat geheugen niet zo maar met een POKE functie informatie kunnen neerzetten. Bij gebruik van POKE is het dus nodig precies te weten waar deze 'gevaarlijke' gebieden zich bevinden. Daarom moet u weten hoe het systeem zelf is opgebouwd. Voorts wordt nadrukkelijk verwezen naar de gebruiksaanwijzing van het systeem. In hoofdstuk 16 is de memory map van de Teleac/DAI personal computer gegeven. De verdeling voor geheugen en I/O is ook daar weergegeven.
Ondanks de gevaren is de POKE functie zeer krachtig. Het is met PEEK en POKE op eenvoudige wijze mogelijk om toch programma's in machinetaal te programmeren. Zo kan de correcte werking van de machine worden getest.
Een belangrijk gebruik van PEEK en POKE bestaat ook uit het overdragen van gegevens vanuit BASIC programma's naar programma's, die direct in de machinetaal zijn geschreven en terug.
In de Teleac/DAI personal computer zoals deze zonder extra uitbreidingen wordt geleverd, is gebruik gemaakt van de memory-mapped-I/O methode. Het systeem heeft daarbij een 8255 general purpose I/O-chip (een algemeen te gebruiken invoer/uitvoerchip) voor de gebruiker ter beschikking. De principe opbouw van de 8255 chip is in hoofdstuk 15 behandeld. Daar is ook de programmering in assembleertaal getoond. Voor details zou u de fabrieksdocumentatie van de 8255 moeten lezen. Wanneer we in BASIC deze chip willen aanspreken, kan dat met de hier behandelde PEEK en POKE functies. De geheugenadressen die voor deze 8255 chip gelden zijn nu in de Teleac/DAI personal computer:

I/O-poort A: adres 65 024 decimaal of FE00 hexadecimaal
I/O-poort B: adres 65 025 decimaal of FE01 hexadecimaal
I/O-poort C: adres 65 026 decimaal of FE02 hexadecimaal
Besturingspoort: adres 65 027 decimaal of FE03 hexadecimaal

De I/O-poorten A, B en C kunnen als input of output worden ingesteld met de besturingspoort. Daarvoor moet een afgesproken code naar deze besturingspoort worden verzonden. Deze code staat in de documentatie van de 8255. Soms worden de poorten A, B en C wel aangeduid met de nummers 0, 1 en 2.
Om een getal uit te sturen op I/O-poort A is dus het volgende programma nodig:

```
*LIST
10    REM VOORBEELD VAN HET GEBRUIK VAN DE ACHTERCONNECTOR
20    POKE 65027,128
30    POKE 65024,0
40    POKE 65024,255
50    POKE 65024,0
*RUN
*
```

DCE-BUS PINOUT

SIGNAL NAME	DESCRIPTION		pin on real-world card	pin on personal computer
P0B0	General Interface PORT 0	Bit 0	24	16
P0B1		Bit 1	26	14
P0B2		Bit 2	28	12
P0B3		Bit 3	30	10
P0B4		Bit 4	29	9
P0B5		Bit 5	27	11
P0B6		Bit 6	25	13
P0B7		Bit 7	23	15
P1B0	General Interface PORT1	Bit 0	12	30
P1B1		Bit 1	10	31
P1B2		Bit 2	8	32
P1B3		Bit 3	7	25
P1B4		Bit 4	9	24
P1B5		Bit 5	11	23
P1B6		Bit 6	13	22
P1B7		Bit 7	15	21
P2B0	General Interface PORT 2	Bit 0	18	26
P2B1		Bit 1	17	27
P2B2		Bit 2	16	28
P2B3		Bit 3	14	29
P2B4		Bit 4	19	20
P2B5		Bit 5	20	19
P2B6		Bit 6	21	18
P2B7		Bit 7	22	17
EXINTR+	External Interrupt		4	6
IN7+	Parallel Input Bit 7 (aux. interrupt)		3	5
EXRESET	External Reset (Ground for Reset)		5	7
+12 V	+12V DC		2	2
+ 5 V	+ 5V DC		31	1
- 5 V	- 5V DC		1	3
GND	Digital and Power Ground		6	4

Afb. 17.4. DCE-Bus

Regel 2Ø stelt de 3 poorten als output in, waarna de regels 3Ø, 4Ø en 5Ø alle bits van I/O-poort A van Ø naar 1 laten gaan en dan weer terug naar Ø.
De aansluitingen van de 8255 zijn naar de achterzijde van de computer uitgevoerd (DCE-bus). U moet dan uiteraard nog weten op welke pennen de verschillende signalen zijn aangesloten. Afbeelding 17.4 geeft de bladzijde uit de computerhandleiding weer waarop dit is aangegeven. De I/O-poorten A, B en C zijn hierin onder de naam General Interface PORT Ø, 1 en 2 aangeduid. De pennummers (PIN) zijn op de connector (stekker) zelf aangegeven. Dus bijvoorbeeld: I/O-poort A bit 5 vindt u op pennummer 11.
Op deze connector zijn ook nog enkele signalen meer uitgevoerd. Zij hebben te maken met RESET van het systeem, met interrupt en met voedingsspanningen. Hierop gaan we niet verder in.
De BASIC functies INP en OUT kunnen hiervoor dus niet gebruikt worden. Het systeem is echter uitbreidbaar met de zgn. DAI Real World Cards. Deze kaarten worden via de 8255 chip met een kabel aan de Teleac/DAI personal computer gekoppeld.
De kaarten moeten worden aangesproken via de INP en OUT functies. In machinetaal programmering is daartoe een tweetraps adressering nodig. Eerst moet de 8255 chip in de Teleac/DAI personal computer goed worden aangesproken en daarna moet de Real World Card worden aangesproken.
Deze twee trappen worden in BASIC verborgen door de functies INP en OUT. De BASIC interpreter zelf zorgt wel voor de juiste adressering. We hebben hier alleen het principe uitgelegd. Voor de details bij deze uitbreiding wordt verwezen naar de handleiding van de Teleac/DAI personal computer en de handleiding, die men krijgt bij aankoop van de Real World Card. Dus voor de standaard Teleac/DAI personal computer mogen de functies INP en OUT niet worden gebruikt! De directe input en output wordt via de PEEK en POKE functies geregeld. Van iedere computer kunnen er verschillende versies in omloop zijn. Wanneer u de computer I/O wilt gebruiken moet u wel eerst in de handleiding controleren of de hier vermelde adressen correct zijn.

We moeten ons realiseren dat voor vrijwel iedere computer de aangesloten I/O-mogelijkheden anders zijn. Gewoonlijk zal het wel mogelijk zijn om zowel parallelle I/O als serie I/O vanuit BASIC en machinetaal te verzorgen. De verschillen zitten voornamelijk in het aantal I/O-lijnen dat naar buiten is uitgevoerd, met de juiste betekenis ervan. Meestal kan dat aantal I/O-lijnen uitgebreid worden door extra aankopen bij de standaard geleverde computer.
De Exidy Sorcerer computer heeft standaard 1 serie I/O ingang en uitgang. Er is 8 bits parallelle I/O plus handshake aanwezig. Uitbreiding is mogelijk via een aparte kast, de S-100 extension unit genoemd.

17.5. VARPTR (X)

Als programmeur in BASIC weet u niet waar de interpreter de gebruikte variabelen precies neerzet. Toch is het in combinatie met de PEEK en POKE functies soms wenselijk de plaats van die variabelen te kennen. Ook hiervoor biedt de Teleac/DAI personal computer in

BASIC een voorziening. Het is de functie VARPTR (X) of VARiable PoinTeR = variabele aanwijzer.
Deze functie geeft als resultaat de decimale waarde van het beginadres, waar de variabele X zich bevindt. De gebruiker moet, om goed van deze functie gebruik te kunnen maken, zelf weten in welk formaat zo'n variabele is opgeslagen. Dat wil zeggen dat hij moet weten hoeveel geheugenplaatsen de variabele beslaat. Daarnaast moet hij weten wat de juiste betekenis van ieder bit op die geheugenplaatsen voor de gelokaliseerde variabele is. Gewoonlijk staat dit in de handleiding van uw systeem aangegeven.

Voorbeeld

```
*LIST
1Ø    REM VOORBEELD VAN VARPTR
2Ø    INPUT "A";A:PRINT
3Ø    LET ADRES=VARPTR(A)
4Ø    FOR I=Ø.Ø TO 3.Ø
5Ø    PRINT PEEK(ADRES+I);
6Ø    NEXT I
7Ø    PRINT
8Ø    IF A<>Ø GOTO 2Ø
9Ø    END
*RUN
A?1
 1 128 Ø Ø
A?2
 2 128 Ø Ø
A?3
 2 192 Ø Ø
A?-1
 129 128 Ø Ø
A?-2
 13Ø 128 Ø Ø
A?.24E14
 45 174 159 124
A?Ø
 Ø Ø Ø Ø
END PROGRAM
*
```

In regel 3Ø zoeken we dus de geheugenpositie van de variabele A op. De regels 4Ø tot en met 6Ø zorgen ervoor dat de geheugeninhoud van 4 plaatsen, waarop we A verwachten, wordt uitgeprint.

17.6. WAIT

Een andere veel voorkomende situatie in I/O-programma's is het wachten op een signaal uit de buitenwereld.
Er wordt dan bijvoorbeeld een opdracht aan de buitenwereld gegeven om iets te doen. Pas als de terugmelding zegt, dat deze opdracht is vervuld, kan het programma zinvol verdergaan.
Dit kan in BASIC worden geprogrammeerd door een loop van PEEK (resp. INP) statements, waarin op de gewenste waarde wordt getest.

Onze BASIC interpreter levert gelukkig een nog prettiger functie daarvoor: de WAIT functie.
De WAIT functie heeft veel mogelijkheden. Die zullen we nu eerst door middel van voorbeelden, oplopend van eenvoudig tot meer complex toelichten.

Voorbeeld

```
*LIST
1Ø    REM WACHT OP HET 1 WORDEN VAN EEN BIT
2Ø    REM UIT DE BUITENWERELD
3Ø    POKE 65Ø27,155
4Ø    LET ADRES=65Ø24.Ø
5Ø    LET MASKER=2.Ø
6Ø    WAIT MEM ADRES,MASKER
7Ø    END
*
```

Op regel 3Ø wordt de 8255 Programmable Peripheral Interface van de besturingspoort (adres 65 Ø27) zo gezet, dat alle I/O-poorten A, B en C inputpoorten zijn. Dit gebeurt door op plaats 65Ø27 de waarde 155 te zetten.
Op regel 6Ø wordt nu vanaf geheugenadres 65 Ø24 een waarde gehaald. Dit geheugenadres correspondeert dus met poort A van de 8255. De EN functie van deze waarde en het *masker* 2 bepaalt of er wordt doorgegaan of niet. Wanneer deze EN een waarde oplevert, die niet nul is, wordt het wachten gestopt en gaan we door naar het volgende statement. Stel dat de opgehaalde waarde 9 decimaal is en ons masker is 2 decimaal:

```
waarde 9 decimaal →  Ø Ø Ø Ø 1 Ø Ø 1
waarde 2 decimaal →  Ø Ø Ø Ø Ø Ø 1 Ø
                     _______________   EN (AND)
                     Ø Ø Ø Ø Ø Ø Ø Ø
```

Het resultaat van de EN functie bestaat slechts uit nullen. Het wachten gaat dus verder. Als daarna de opgehaalde waarde is veranderd tot:

```
waarde 11 decimaal → Ø Ø Ø Ø 1 Ø 1 1
waarde  2 decimaal → Ø Ø Ø Ø Ø Ø 1 Ø
                     _______________   EN (AND)
                     Ø Ø Ø Ø Ø Ø 1 Ø
```

wordt het wachten beëindigd.

Let wel. Via de memory-mapped-I/O-methode van de Teleac/DAI personal computer zal de geheugenplaats een inputpoort functie moeten hebben, zodat de buitenwereld inderdaad bits kan doen veranderen.
Op deze wijze is het ook mogelijk dat één signaal van buiten het wachten beëindigt. Wanneer we bijvoorbeeld als masker de waarde 255 kiezen, zal ieder binnenkomend bit dat hoog is, de wachtsituatie opheffen:

```
waarde  11 decimaal → Ø Ø Ø Ø 1 Ø 1 1
waarde 255 decimaal → 1 1 1 1 1 1 1 1
                      _______________   EN (AND)
                      Ø Ø Ø Ø 1 Ø 1 1
```

Resultaat: we stoppen met wachten.
Het is niet alleen wenselijk te wachten op het hoog (1) worden van een signaal uit de buitenwereld, maar ook op het laag (Ø) worden ervan.
De WAIT functie stelt ons hiertoe in staat.

Voorbeeld

```
*LIST
1Ø    REM WACHT OP HET Ø WORDEN VAN EEN BIT
2Ø    REM UIT DE BUITENWERELD
3Ø    POKE 65Ø27,155
4Ø    LET ADRES=65Ø24.Ø
5Ø    LET MASKER=2.Ø
6Ø    LET OMKEER=255.Ø
7Ø    WAIT MEM ADRES,MASKER,OMKEER
8Ø    END
*RUN
END PROGRAM
*
```

Nu wordt in het WAIT statement op regel 7Ø de waarde, die de buitenwereld op adres 6ØØ24 (I/O-poort A) aanbiedt, ingelezen. Van deze waarde wordt de Exclusieve OF genomen met de waarde van OMKEER. Het resultaat daarvan is dat de waarden van ieder bit geïnverteerd worden, want alle bitwaarden van OMKEER zijn 1. (Dus ieder Ø wordt een 1, terwijl iedere 1 een Ø wordt.)
Tenslotte wordt het resultaat daarvan door middel van de EN functie met de waarde van het masker verwerkt. Wanneer we de ingelezen waarde op 11 decimaal stellen, is het resultaat:

```
ingelezen waarde 11 decimaal →   Ø Ø Ø Ø 1 Ø 1 1
omkeerwaarde 255 decimaal →      1 1 1 1 1 1 1 1
                                 _______________ EX OF (EXOR)
                                 1 1 1 1 Ø 1 Ø Ø
maskerwaarde 2 decimaal →        Ø Ø Ø Ø Ø Ø 1 Ø
                                 _______________ EN (AND)
                                 Ø Ø Ø Ø Ø Ø Ø Ø
```

Het wachten duurt dus voort. Veronderstel dat daarna de ingelezen waarde 9 decimaal wordt, dan verloopt de situatie aldus:

```
ingelezen waarde 9 decimaal →    Ø Ø Ø Ø 1 Ø Ø 1
omkeerwaarde 255 decimaal →      1 1 1 1 1 1 1 1
                                 _______________ EX OF
                                 1 1 1 1 Ø 1 1 Ø
maskerwaarde    2 decimaal →     Ø Ø Ø Ø Ø Ø 1 Ø
                                 _______________ EN
                                 Ø Ø Ø Ø Ø Ø 1 Ø
```

Het wachten wordt nu beëindigd en het programma zal op regel 8Ø verdergaan. We hebben dus hier gewacht op het laag (Ø) worden van een binnenkomend signaal.
Zo is het mogelijk door handig kiezen van de OMKEER- en MASKER-waarde te wachten tot één uit meerdere bits hoog (1) of laag is.
In het volgende wordt een voorbeeld gegeven van wachten tot één

van de linker drie bits laag (Ø) wordt of tot één van de rechter twee bits hoog (1) wordt op adresplaats 2ØØ:

1Ø WAIT MEM 200, 128+64+32+2+1, 128+64+32

Stel dat de binnenkomende waarde 129 decimaal is, dan krijgen we het volgende beeld:

```
ingelezen waarde 129 decimaal → 1 Ø Ø Ø Ø Ø Ø 1
omkeerwaarde 224 decimaal →     1 1 1 Ø Ø Ø Ø Ø
                                ─────────────── EX OF
                                Ø 1 1 Ø Ø Ø Ø 1
maskerwaarde 227 decimaal →     1 1 1 Ø Ø Ø 1 1
                                ─────────────── EN
                                Ø 1 1 Ø Ø Ø Ø 1
```

Het wachten stopt nu omdat aan één van de twee eisen is voldaan. Het voldoet namelijk ofwel aan de eerste eis: één van de linker drie bits is laag (Ø) ofwel aan de tweede eis: één van de rechter twee bits is hoog (1).
In feite geeft u dus met behulp van de OMKEER-waarde aan welke bits op Ø worden getest (die bits worden in de omkeerwaarde op 1 gezet) en welke op 1 (een Ø in de omkeerwaarde). Met het masker bepaalt u daarna welke bits worden getest (op die plaats wordt een 1 gezet).
U ziet dat er met dit WAIT statement erg veel mogelijk is. Maar er zijn meer mogelijkheden. In de bovenstaande voorbeelden gingen we uit van de memory-mapped-I/O methode, dat wil zeggen van de functie:

WAIT MEM I, J, K

Het is ook mogelijk gebruik te maken van een functie WAIT I, J, K. Deze doet hetzelfde als WAIT MEM I, J, K, maar dan gericht op de I/O-poorten van de Real World Card, die als uitbreiding van de Teleac/DAI-computer kunnen worden gekocht. Hier geldt hetzelfde verschil tussen WAIT MEM en WAIT als bij de PEEK en INP functie en bij de POKE en OUT functie. Samenvattend kennen we nu de volgende functies in algemene vorm:

WAIT MEM I, J [,K]
WAIT I, J [,K]

K is de omkeerwaarde die een 'exclusieve of' vormt met de binnengekomen waarde. K moet altijd groter of gelijk aan Ø zijn en kleiner of gelijk aan 255. K mag ook worden weggelaten, hetgeen betekent dat de 'exclusieve of' wordt weggelaten. Voor ieder bit dat op nul getest moet worden, dient het corresponderend bit in K de waarde 1 te hebben.
J is het masker, waarmee de binnengehaalde waarde (die eventueel eerst met K een Exclusieve Of is aangedaan) een logische EN functie ondergaat. J moet altijd kleiner of gelijk aan 255 zijn en groter of gelijk aan Ø. Voor ieder bit dat we willen testen, moet het corresponderende bit in J de waarde 1 hebben.
I is in het geval van WAIT MEM het geheugenadres, vanwaar de waarde wordt ingelezen (memory-mapped-I/O). I moet groter of gelijk aan Ø zijn en kleiner of gelijk aan 65 535. In het geval WAIT is I het nummer van de input-poort van de Real World Card, waarvan

de waarde wordt ingelezen. I moet dan groter of gelijk aan Ø en kleiner of gelijk aan 255 zijn.
In bovenstaande voorbeelden wordt er steeds gewacht totdat een van buiten komend signaal hoog of laag wordt. In veel toepassingen moet er echter gewoon een bepaalde tijd worden gewacht. Ook dat is in BASIC mogelijk bij onze Teleac/DAI personal computer. De functie, die daarvoor ter beschikking staat, is WAIT TIME.

Voorbeeld

```
1Ø  REM VOORBEELD VAN EEN TIJDJE WACHTEN
2Ø  LET ZOLANG = 1ØØØ
3Ø  WAIT TIME ZOLANG
4Ø  PRINT "KLAAR"
```

In dit voorbeeld wachten we 2Ø seconden, namelijk 1ØØØ × 2Ø milliseconden. De functie in algemene vorm is:

WAIT TIME I

I geeft daarbij de tijd aan 2Ø milliseconden, dat er gewacht moet worden. In het begin van dit hoofdstuk zagen we bij het looplicht hoe een vertragingsloop werd ingebouwd. Iets dergelijks kan natuurlijk ook met het WAIT statement gebeuren. Daarom voegen we hier een voorbeeld van gebruik bij het looplicht toe.

Voorbeeld

```
*LIST
5     PRINT "TYPE EEN GETAL"
6     PRINT "ER ZAL DIT GETAL KEER 2Ø MS GEWACHT WORDEN"
7     PRINT "BIJ HET LOPEN VAN DE LAMP"
8     INPUT GETAL
1Ø    REM VERTRAGING LOOPLICHT OP OUTPUTPOORT 1
2Ø    REM MET GEBRUIK VAN WAIT TIME
3Ø    LET ADRES=1.Ø
4Ø    FOR I=Ø.Ø TO 7.Ø
5Ø    LET WAARDE=1.Ø SHL I
6Ø    OUT ADRES,WAARDE
62    WAIT TIME GETAL
7Ø    NEXT I
8Ø    GOTO 4Ø
*
*
*
*
*RUN
TYPE EEN GETAL
ER ZAL DIT GETAL KEER 2Ø MS GEWACHT WORDEN
BIJ HET LOPEN VAN DE LAMP
?2
```

Het is tenslotte nuttig erop te wijzen dat de ingangs- en uitgangssignalen van de computer digitale signalen zijn. Deze zijn niet ogenblikkelijk gebruiksklaar om bij voorbeeld met een lamp of schake-

laar te verbinden. Er moet nog een aanpassing tussen. De gewenste aanpassing kan van verschillende aard zijn. In *Microprocessors 1* is hier reeds op ingegaan. Op deze aanpassing dieper ingaan zou ons te ver in de techniek voeren.
In *Microprocessors 1* is genoemd dat voor I/O soms D.M.A. (Direct Memory Acces) nodig is. In BASIC zijn geen bijzondere functies aanwezig, die het werk met interrupt of D.M.A. verzorgen. Deze methoden moeten dus via machinetaal (geprogrammeerd via de assembler) worden afgehandeld.
Niet iedere versie BASIC zal zulke uitgebreide I/O faciliteiten bieden als hier behandeld zijn. Met name bij BASIC op grote computers is meestal PEEK/POKE/WAIT niet aanwezig. Dit is om te voorkomen dat de programmeur rechtstreeks vanuit BASIC het computersysteem beïnvloedt en daarbij andere programmeurs in moeilijkheden brengt, of zelfs het hele systeem in verwarring brengt. Vooral de POKE is dan de boosdoener. Een BASIC-programmeur op zo'n groot systeem mag alle adressen in geheugen en I/O gewoonlijk dan ook niet kennen.
Ook hier is er dus geen echte standaard aanwezig. Opnieuw hebben we als uitgangspunt een uitgebreide versie van BASIC verkozen.

Samenvatting hoofdstuk 17

In dit hoofdstuk hebben we gezien welke directe I/O-mogelijkheden de Teleac/DAI personal computer via de BASIC functies biedt. We zagen daarbij dat de standaard I/O op basis van de memory-mapped-methode is gebaseerd. Met behulp van PEEK en POKE kunnen we I/O poorten bereiken. De plaats van een BASIC variabele wordt met behulp van het VARPTR statement gevonden.
Tenslotte zijn er uitgebreide mogelijkheden besproken om te wachten op het hoog of laag worden van een signaal uit de buitenwereld in de functie WAIT. Met deze functie kan tevens een van tevoren bepaalde tijd gewacht worden.

Multiple choice-vragen bij hoofdstuk 17

A. Het statement LET B = INP(I) haalt in de standaard Teleac/DAI personal computer:

1 de waarde I van inputpoort B
2 de waarde B van geheugenplaats I
3 een waarde van inputpoort B en zet deze in I
4 een waarde van inputpoort I en kent deze toe aan de variabele B.

B. Bij memory-mapped I/O

1 worden I/O poorten als aparte apparatuur beschouwd
2 wordt geheugen als deel van I/O poorten beschouwd
3 worden I/O poorten als deel van het geheugen beschouwd
4 kan I/O in BASIC gebruikt worden

C. Bij gebruik van het statement POKE I, J moet gelden dat:

1 I loopt van 0 t/m 65535, terwijl J loopt van 0 t/m 255
2 J loopt van 0 t/m 65535, terwijl I loopt van 0 t/m 255
3 de gebruikte geheugenplaats ook echt aanwezig is
4 de gebruikte I/O poort ook echt aanwezig is

D. a. I/O-poorten kan men in de standaard Teleac/DAI personal computer via INP(X) en OUT I, J gebruiken.
b. INP(X) en OUT I, J worden gebruikt voor memory-mapped-I/O.

1 uitspraak a is goed en uitspraak b is goed
2 uitspraak a is fout en uitspraak b is goed
3 uitspraak a is goed en uitspraak b is fout
4 uitspraak a is fout en uitspraak b is fout

E. a. Met het statement WAIT, I, J kan een outputpoortwaarde getest worden.
b. Met het statement WAIT I, J kan tijd afgeteld worden.

1 uitspraak a is goed en uitspraak b is goed
2 uitspraak a is fout en uitspraak b is goed
3 uitspraak a is goed en uitspraak b is fout
4 uitspraak a is fout en uitspraak b is fout

F. a. WAIT kan zowel op directe I/O als op memory-mapped I/O werken.
b. WAIT heeft maximaal 3 argumenten

1 uitspraak a is goed en uitspraak b is goed
2 uitspraak a is fout en uitspraak b is goed
3 uitspraak a is goed en uitspraak b is fout
4 uitspraak a is fout en uitspraak b is fout

G. Wanneer in het voorbeeldprogramma: wacht op het 1 worden van een bit uit de buitenwereld, in regel 50, het masker op 0 (nul) wordt gezet in plaats van 2 dan wordt er:

1 gewacht op het 0 worden van alle bits van de inputpoort
2 eindeloos gewacht
3 gewacht op het 1 worden van alle bits van de inputpoort
4 nooit gewacht

H. a. VARPTR (B) geeft als resultaat de hexadecimale waarde van het beginadres van B in het geheugen.

b. VARPTR (B) geeft als resultaat de decimale waarde van het eindadres van B in het geheugen.

1 uitspraak a is goed en uitspraak b is goed
2 uitspraak a is fout en uitspraak b is goed
3 uitspraak a is goed en uitspraak b is fout
4 uitspraak a is fout en uitspraak b is fout

I. a. Het BASIC statement OUT I, J zal langzamer worden uitgevoerd dan de vergelijkbare machinetaal instructie.

b. In BASIC kunnen niet alle I/O-poorten worden bediend, terwijl dit in machinetaal wel mogelijk is.

1 uitspraak a is goed en uitspraak b is goed
2 uitspraak a is fout en uitspraak b is goed
3 uitspraak a is goed en uitspraak b is fout
4 uitspraak a is fout en uitspraak b is fout

J. a. Met een POKE I, J-statement kan de correcte werking van de computer verstoord worden.

b. INP(X) en OUT I, J zijn vooral belangrijk voor het overdragen van gegevens tussen een BASIC programma en machinetaal programma's.

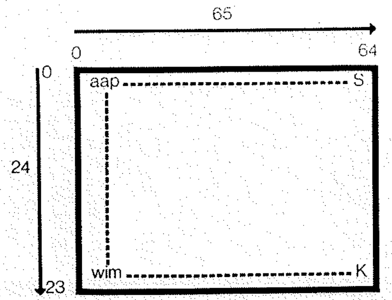
1 uitspraak a is goed en uitspraak b is goed
2 uitspraak a is fout en uitspraak b is goed
3 uitspraak a is goed en uitspraak b is fout
4 uitspraak a is fout en uitspraak b is fout

Hoofdstuk 18

BASIC programmering voor beeld en geluid

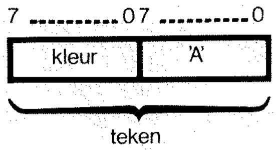
In dit hoofdstuk zullen we de speciale mogelijkheden voor beeld en geluid, waarover de Teleac/DAI-computer beschikt, behandelen. Deze mogelijkheden kunnen zowel in BASIC als in 8080 machinetaal worden gebruikt. Veel personal computers beschikken over vergelijkbare mogelijkheden.

18.1. Beeldopbouw



Afb. 18.1. Beeldscherm 24 regels van 64 tekens

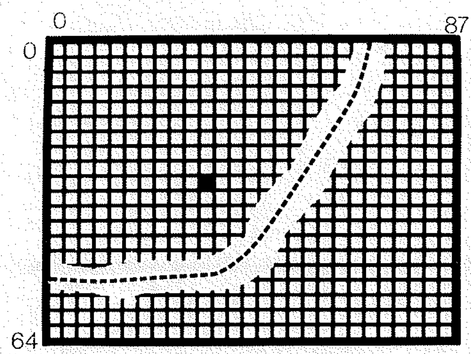
We bekijken hoe het beeld op het beeldscherm tot stand komt. Het scherm kan uiteraard in de eerste plaats worden gebruikt om er tekst op zichtbaar te maken.
De tekst, die we zichtbaar maken, bevat maximaal 24 regels van 64 tekens (zie afbeelding 18.1).
Voor het opbouwen van het videobeeld beschikt de Teleac/DAI-computer over speciale apparatuur, die we in het vervolg de *videogenerator* noemen. Deze haalt de informatie die nodig is voor het beeldscherm, uit het RAM geheugen.
Wanneer een door de programmeur gewenst beeld op het TV-scherm moet komen, moeten we dat in afgesproken geheugenplaatsen neerzetten. De videogenerator zorgt er dan voor dat deze informatie op het beeldscherm verschijnt. Van het in de computer aanwezige RAM geheugen wordt dus een deel gebruikt voor de beeldopbouw. In dit gedeelte kunnen we uiteraard geen gewone instructies schrijven. Het beeld zou dan namelijk verstoord worden. Daarom staat er voor het programma minder geheugen ter beschikking dan er totaal aan geheugen aanwezig is.
De standaard Teleac/DAI personal computer kan alleen een zwart/wit beeld oproepen. De mogelijkheid bestaat deze computer met kleur uit te breiden. In dat geval kunnen we 16 verschillende kleuren gebruiken. Ieder van die kleuren kan als achtergrond- of voorgrondkleur fungeren. Wanneer we denken aan zwart getypte letters op een wit vel papier, noemen we zwart de voorgrond- en wit de achtergrondkleur.



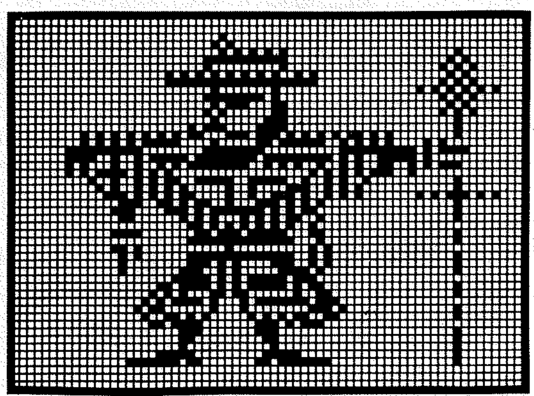
Afb. 18.2. Tekenopbouw voor het beeldscherm in geheugen

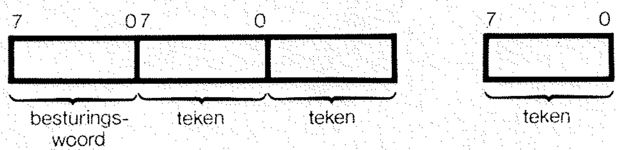
Voor de aanduiding van de tekens gebruiken we de ons reeds bekende ASCII code (zie hoofdstuk 10), die in feite uit 7 bits bestaat. Het 8e bit in een byte waarin de ASCII code staat, blijft hier ongebruikt. Wanneer we nu ook nog de kleur van een letter willen aangeven, is het duidelijk, dat we meer bits moeten toevoegen aan de ASCII code. In onze computer wordt ieder teken voor het

beeldscherm dan ook opgebouwd uit 2 bytes (16 bits). Het laagste orde bevat de ASCII code van het gewenste teken en het hoger orde byte, de kleur informatie (zie afbeelding 18.2).
We zijn echter niet alleen in staat tekst op ons scherm te doen verschijnen. We kunnen ook plaatjes oproepen. Ieder plaatje wordt daarbij opgebouwd uit beeldpunten. Een beeldpunt is een rechthoekig vlakje op het TV-scherm. De programmeur kan daarbij weer kiezen uit het aantal beeldpunten, waaruit hij het beeld wil laten bestaan. De keuzemogelijkheden zijn hier in principe 65 verticaal bij 88 horizontaal hetzij 130 bij 176 of 260 bij 352 beeldpunten (zie afbeelding 18.3).
Wanneer we ons een zwart-wit beeld voorstellen, kan de programmeur van ieder beeldpunt aangeven of het zwart dan wel wit moet zijn. Op deze wijze kan een plaatje worden getekend (zie afbeelding 18.4). Amerikanen noemen deze beeldpunten wel PIXELS (PIXure ELementS).
Wanneer we over de mogelijkheid van kleurtoepassing beschikken kunnen we in principe ieder vlakje een van de 16 kleuren laten aannemen. Dit geeft een geweldige variëteit aan mogelijke figuren. Afbeelding 18.5 geeft een regel tekst in het geheugen weer.



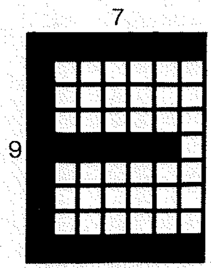
Afb. 18.3. Beeldopbouw in grafische mode 56 × 88


Afb. 18.4. Plaatje opgebouwd uit beeldpunten

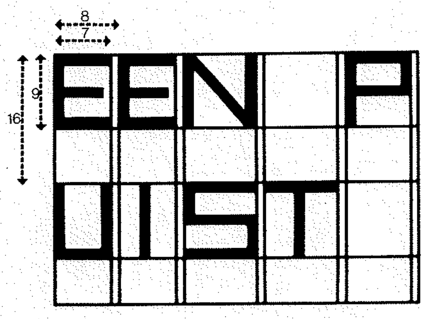


Afb. 18.5. Regelopbouw voor het beeldscherm in het geheugen

De videogenerator moet uiteraard wel weten of het beeld als plaatje (*grafische mode*) of als gewone tekst (*charactermode*) moet worden opgebouwd. Iedere regel in het geheugen voor de beeldopbouw wordt daarom nog door besturingsgegevens voor de videogenerator voorafgegaan. Deze besturingsgegevens beslaan eveneens 2 bytes. We duiden deze bytes in het vervolg aan met de term besturingswoord. Het besturingswoord geeft dus aan welke grafische opbouw moet worden gebruikt en wat de achtergrondkleur van het beeld wordt.



Afb. 18.6. Een letter opgebouwd uit 7 × 9 punten



Afb. 18.7. Beeldopbouw met letters

Code	Colour
0	Black
1	Blue
2	Petunia Red
3	Brown
4	Storm Blue
5	Green
6	Kakhi Brown
7	Orange
8	Gray
9	Lavender
10	Cantalope Yellow
11	Pink
12	Sky Blue
13	Pea Green
14	Yellow
15	White

Afb. 18.8. Nummers met de voor de kleur (van het kleurenbeeldscherm) gekozen benaming (kleurentabel)

Voor een beter begrip verdiepen we ons eerst in de manier waarop tekens worden opgebouwd. De vorm van een teken zoals dat op het scherm moet verschijnen, wordt bepaald met behulp van een zogenaamde *character generator*. Deze is in feite niets anders dan een ROM, waarin de ASCII code in een aantal punten wordt vertaald, die op het beeldscherm moeten verschijnen. Ieder teken wordt als een matrix van 7 × 9 punten afgebeeld (zie afbeelding 18.6).
De character generator houdt rekening met de ruimtes, die horizontaal en verticaal tussen tekens moeten worden aangehouden. In feite geeft hij daarom een patroon van 8 × 16 punten af (zie afbeelding 18.7).
Met behulp van een andere character generator kan dus in principe een andere vorm van de characters worden gekozen, zoals we bij sommige computers in deze les ook zullen zien.
Voor de grafische mode is het beeld uiteraard veel eenvoudiger. Het hele vlak krijgt daar de gewenste kleur. De vorm is gewoon een rechthoekje, dat geheel moet worden opgevuld. De vlakjes grenzen aan elkaar, zodat doorlopende lijnen gemaakt kunnen worden.
Een TV-toestel bouwt een beeld uit beeldlijnen op (op een Europees TV-toestel 625 beeldlijnen). De videogenerator moet steeds zorgen dat in iedere lijn het juiste gedeelte van de characters wordt afgebeeld. Die gedeelten haalt de videogenerator uit de character generator. Op een volgende beeldlijn wordt dan het volgende stukje van de characters afgebeeld, enz.
Is een hele regel tekst opgebouwd, dan wordt op enige beeldlijnen geen characterinformatie gezet, zodat tussen de tekstregels ruimte ontstaat. Deze ruimte wordt weer bepaald door de character generator zoals afbeelding 18.7 al aangaf.
Tenslotte wordt het signaal van de videogenerator nog geschikt gemaakt voor invoer via de antenne-ingang van de TV. Daarvoor is een zogeheten modulator nodig. Een programmeur kan dus door middel van een in machinetaal geschreven programma precies bepalen wat het beeldscherm te zien zal geven. Hij vult daarvoor het besturingswoord en de daarop volgende tekens van iedere regel met de door hem gewenste informatie.
Hij moet dan uiteraard tot in alle details op de hoogte zijn van de mogelijke inhoud van besturingswoord en tekens. Om dat goed te kunnen is vrij veel detailkennis van het systeem nodig. De handleiding bij de computer dient deze kennis aan te geven.
In BASIC is het mogelijk om met behulp van POKE statements hetzelfde resultaat te bereiken. Men moet dan evenwel ook over de gedetailleerde systeemkennis beschikken. Vandaar dat er in BASIC ook eenvoudiger manieren zijn om het beeld naar wens samen te stellen, wat nu behandeld wordt.

18.2. Beeld met BASIC

De kleuren die we met de Teleac/DAI personal computer kunnen produceren, zullen we in de rest van dit hoofdstuk met nummers aangeven. Voor de 16 mogelijke kleuren lopen de nummers van Ø tot en met 15.
Afbeelding 18.8 geeft bij ieder nummer de kleur aan zoals deze op het kleurenbeeldscherm verschijnt, met de voor de kleur gekozen benaming.

Bezitters van een computer in zwart/wit uitvoering zullen deze kleuren als verschillende grijsgradaties terugvinden. Deze gradaties lopen van wit (15) tot zwart (Ø).
Hetzelfde verschijnsel doet zich voor bij een kleurencomputer met een zwart/wit TV-toestel.
Het eerste BASIC statement dat we behandelen, is het statement COLORT. Dit statement biedt de mogelijkheid de voor- en achtergrondkleur van de tekst (vandaar COLORT = kleur tekst) op het beeldscherm te kiezen.

Voorbeeld

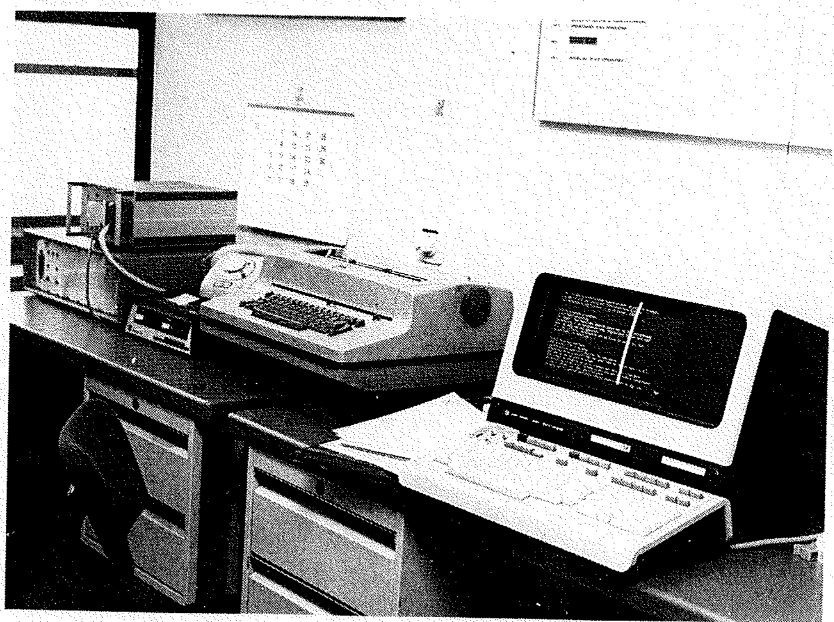
COLORT 1 4 7 14

Na uitvoering van dit statement zal alle tekst op het scherm in kleur 1 als voorgrond en in kleur 4 als achtergrond verschijnen. De kleuren 7 en 14 worden voor in machinetaal geschreven programma's respectievelijk als voorgrond en als achtergrond ter beschikking gesteld.
Voor het statement COLORG geldt eenzelfde redenering, maar dan voor de grafische modes (G = grafisch).

Voorbeeld

COLORG 5 9 3 7

De kleuren, die in een 4-kleuren grafische mode (zie voor uitleg verderop in dit hoofdstuk) gebruikt kunnen worden, zijn na uitvoering van dit statement: 5, 9, 3 en 7.
De eerste kleur (hier 5) geeft tevens aan dat van de 16 kleuren grafische mode de algemene achtergrondkleur groen wordt. Verdere beïnvloeding van de 16 kleuren grafische mode treedt niet op.
De statements COLORG en COLORT beïnvloeden elkaars kleuren niet. Dat wil zeggen dat we de kleuren, gebruikt in de karaktermode (tekst), afzonderlijk dienen te zien van de kleuren die we in de grafische mode gebruiken (plaatjes).



Tekstverwerkingssysteem vakgroep Digitale Techniek TH Twente.

Verder staat ons in BASIC het statement MODE ter beschikking. Hiermee stellen we in feite de reeds behandelde mogelijkheden voor indeling van het beeld in.
Vanuit BASIC kunnen we zo 7 verschillende modes (= beeldgebruik) instellen, genummerd van Ø tot en met 6. Deze zijn achtereenvolgens:

mode Ø karaktermode tekst van 24 regels bij 64 karakters
mode 1 16 kleuren grafische mode in 65 × 88 beeldpunten
mode 2 4 kleuren grafische mode in 65 × 88 beeldpunten
mode 3 16 kleuren grafische mode in 13Ø × 176 beeldpunten
mode 4 4 kleuren grafische mode in 13Ø × 176 beeldpunten
mode 5 16 kleuren grafische mode in 26Ø × 352 beeldpunten
mode 6 4 kleuren grafische mode in 26Ø × 352 beeldpunten

Voor de mode 1 tot en met 6 is het mogelijk om een A toe te voegen (bij voorbeeld MODE 1A). Het onderste gedeelte van het beeld kan dan worden gebruikt om 4 regels tekst neer te zetten. Dat wil zeggen, dat we op deze manier een plaatje met onderschrift construeren (zie afbeelding 18.9).



Afb. 18.9. Tekst + grafische mode

Het gebruik van het RAM geheugen voor de beeldopbouw is steeds zodanig dat zo weinig mogelijk RAM geheugen van de programmeur wordt afgenomen. In iedere mode beziet de Teleac/DAI personal computer dan ook hoeveel RAM geheugen voor het beeld nodig is. Met name in de grafische modes is een zo handige opbouw gekozen dat meer beeldpunten in de 2 bytes voor de tekens van het beeld worden gebruikt. Dit is gedaan om beeldruimte in RAM te besparen. Om de besparingsmogelijkheid nog groter te maken is er ook een 4-kleuren grafische mode geïntroduceerd. De kleurcodering beslaat dan slechts 2 bits (voor 4 kleuren) in plaats van 4 bits (voor 16 kleuren). Zo kan het beeld nog compacter in het geheugen worden gezet. Voor veel eenvoudige figuren is het namelijk voldoende om maar met 4 kleuren te werken.
Preciese details van het geheugengebruik horen in de handleiding van de computer te zijn aangegeven. Voor de charactermode kunnen we eenvoudig berekenen dat voor iedere regel nodig is: 64 tekens en 1 besturingswoord van ieder 2 bytes, dus 13Ø bytes. Er zijn 24 regels, dus in totaal zijn er 312Ø bytes nodig.
Het is een bekend verschijnsel dat het beeld aan de horizontale zijden van een TV-beeldscherm niet geheel zichtbaar is. De

programmeur kan daarmee rekening houden door in horizontale richting het scherm met 16 beeldpunten minder te beschrijven; dus aan beide zijden 8 beeldpunten blanco laten. De gebruikte grafische modes worden in dit geval 65 x 72; 13Ø x 16Ø en 26Ø x 336.
Het is eveneens een bekend verschijnsel dat een zwart-wit computer aangesloten op een kleuren TV niet exact de kleuren zwart en wit geeft. Er wordt meestal iets kleur zichtbaar. Hiervan hoeft u niet te schrikken, uw computer zowel als uw TV zijn in orde. U kunt dit probleem niet oplossen.

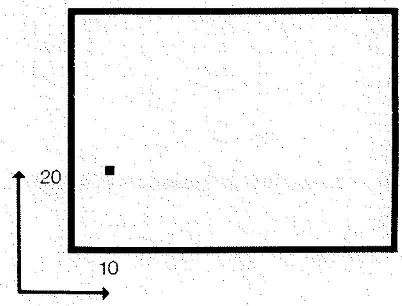
18.3. Tekenen in BASIC

Nadat we in BASIC de verschillende beeldopbouwmogelijkheden hebben kunnen vastleggen, willen we ook nog over de mogelijkheid beschikken om direct in BASIC beeldpunten op het scherm in te vullen. Daarvoor bestaat het statement DOT.

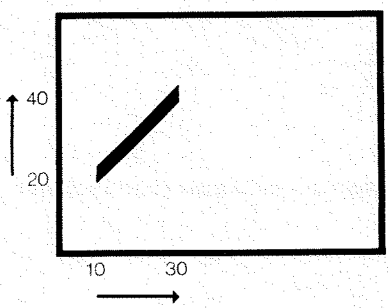
Voorbeeld

DOT 1Ø, 2Ø Ø

Het beeldpunt op de 1Øe regel van het scherm en de 2Øe horizontale beeldpositie krijgt de kleur 15. Waar de 1Øe regel en het 2Øe beeldpunt zich bevinden, wordt bepaald door de grafische mode, die van tevoren is ingesteld (zie afbeelding 18.10).



Afb. 18.10. DOT 10,20, 0



Afb. 18.11. DRAW 10,20 30,40 0

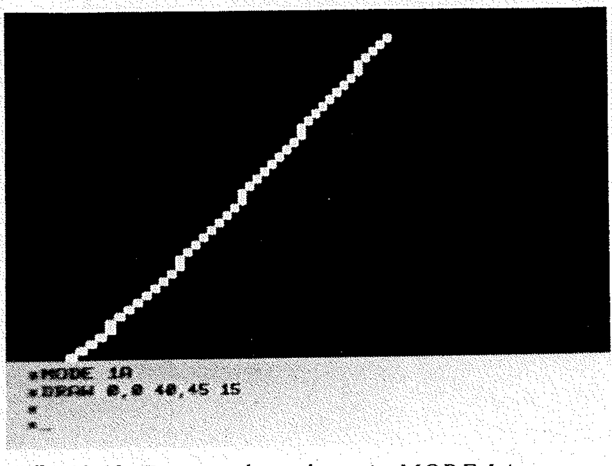
Bij het maken van eenvoudige plaatjes komen rechte lijnen nogal eens voor. Vandaar dat de Teleac/DAI personal computer ook daarin voorziet door middel van het statement DRAW.

Voorbeeld

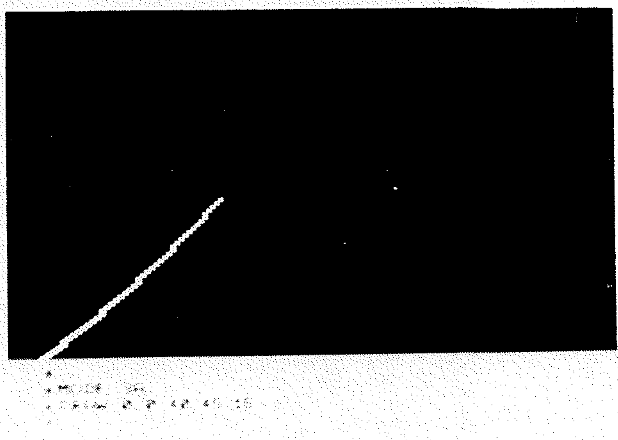
DRAW 1Ø, 2Ø 3Ø, 4Ø Ø

Er wordt nu een lijn getrokken tussen het beeldpunt op de 1Øe regel, de 2Øe horizontale positie en het beeldpunt op de 3Øe regel, 4Ø horizontale beeldpositie.
De kleur van deze lijn is 15 (zie afbeelding 18.11).
Sommige computers beschikken over de mogelijkheid om lijnen langs meer punten te laten trekken. Daar geeft men dan meer dan 2 punten op. De lijn wordt van het eerste naar het tweede punt getrokken, dan van het tweede naar het derde, enz.
Dit vereenvoudigt het maken van plaatjes aanzienlijk. Men moet zich uiteraard wel realiseren dat het beeld nog steeds uit een beperkt aantal beeldpunten is opgebouwd. Vooral in de grafische mode van

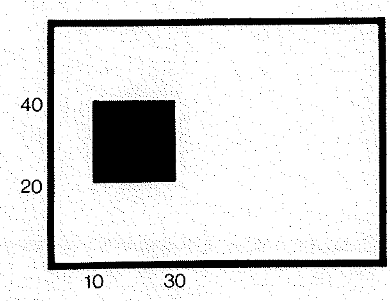
65 × 88 zal deze rechte lijn een herkenbaar trappenpatroon vertonen. Hoe meer beeldpunten het scherm kan bevatten des te beter kan de tekening op het scherm een echte tekening zoals op papier benaderen (zie afbeelding 18.12 en 18.13).



Afb. 18.12. Foto van het scherm in MODE 1A met DRAW 0,0 40,45 15



Afb. 18.13. Foto van het scherm in MODE 3A



Afb. 18.14. FILL 10,20 30,40 0

De laatste mogelijkheid die in de grafische mode gebruikt kan worden, is het statement FILL. Dit statement vult een vlak tussen twee punten. Daartoe beschouwt het de twee punten als 2 hoekpunten van een rechthoek. De zo ontstane rechthoek wordt opgevuld met een gespecificeerde kleur (zie afbeelding 18.14 voor FILL 1Ø,2Ø 3Ø,4Ø Ø). Op deze wijze hebben we een behoorlijk krachtig pakket statements DOT, DRAW en FILL om eenvoudige plaatjes op het beeldscherm te maken.
Deze statements werken vanzelfsprekend alleen in de grafische modes. In de charactermode (mode Ø) zullen zij aanleiding geven tot een foutmelding: OFF SCREEN.
Met bovengenoemde statements kunnen we in feite allerlei figuren op het scherm construeren. Om dit nog gemakkelijker te maken zijn er nog extra statements aanwezig. Deze zijn XMAX, YMAX en SCRN (X,Y). XMAX geeft als resultaat de maximum waarde voor X in de huidige ingestelde mode. Dus:

```
2Ø A = XMAX
```

geeft aan A de maximale toegestane waarde van X. In mode 1 bijvoorbeeld is dit dus 87. Zo geeft YMAX de maximale waarde van Y. Programma's kunnen op deze wijze algemeen geschreven worden zonder dat uitgebreid rekening moet worden gehouden met de mode die gebruikt is. De mode kan via XMAX en YMAX worden verrekend.
SCRN (X,Y) geeft aan welke kleur op het scherm op positie X,Y aanwezig is. Het BASIC programma hoeft dan niet steeds te onthouden. Het vraagt dit op met SCRN (X,Y).

Voorbeeld

```
2Ø A = SCRN (3Ø,4Ø)
```

geeft aan A de waarde van de kleur op X-positie 3Ø en Y-positie 4Ø.

We kunnen uiteraard ook alleen met tekst op het scherm werken (mode Ø). Voor veel toepassingen is het dan wenselijk om de *cursor* naar een gewenste positie te brengen zonder de reeds bestaande tekst te veranderen. Denk daarbij aan het invullen van een op het scherm afgedrukt formulier. Ook daarvoor heeft de Teleac/DAI personal computer faciliteiten in BASIC. Het betreft hier de de statements CURSOR (X,Y), CURX en CURY.
Met CURSOR (X,Y) wordt de cursor (het knipperende vierkantje) gezet op positie X,Y. Hierbij is Y het regelnummer van onder af aan op het scherm te rekenen.
X is de positie in de regel waarbij positie Ø zich links bevindt.
Met CURX en CURY is het mogelijk de huidige positie van de cursor op te vragen in X-, respectievelijk Y-richting. Zo kan dus ook met tekst worden gemanipuleerd.
Bij de bovenstaande beschouwingen moeten we wel bedenken dat gewone printers niet in staat zijn om de getekende plaatjes af te drukken. Deze zijn slechts afgestemd op afdrukken van tekst. Wil men toch deze plaatjes afdrukken dan is een fototoestel ofwel een heel speciale printer nodig. Deze laatste printers worden wel aangeduid met de naam *graphic printers* (grafische printers).

18.4. Geluid

We hebben nu gezien dat de Teleac/DAI personal computer over veel mogelijkheden beschikt om het scherm voor tekst en plaatjes te gebruiken. Daarbij is ook aangegeven hoe daarbij van kleuren gebruik wordt gemaakt. In een zwart/wit uitvoering zullen in plaats van kleuren tinten grijs verschijnen. Maar, het blijft ook dan nog steeds goed mogelijk om plaatjes te construeren.
In dit hoofdstuk zullen we nu kort op een andere mogelijke uitbreiding van de Teleac/DAI personal computer ingaan, namelijk het programmeren van geluid. Dit kan zelfs in stereo geschieden. Zoals het beeld wordt opgewekt met een apart hardware-gedeelte, dat videogenerator heet, zo wordt het geluid opgewekt door middel van een *soundgenerator*. Deze is als optie verkrijgbaar.
Geluid wordt geproduceerd als trilling en met een bepaalde frequentie. Deze trillingen kunnen in de computer op digitale wijze worden opgewekt door middel van programmeerbare oscillatoren. Een *oscillator* kan men zich voorstellen als een in trilling gebrachte stemvork. Het programmeerbare heeft betrekking op de frequentie van de stemvork (ofwel de toon), die via een programma kan worden ingesteld. Onze computer bevat 3 verschillende oscillatoren, die ieder onafhankelijk van elkaar de tonen produceren. Bovendien is in het systeem nog een ruisgenerator aanwezig waarvan, populair gezegd, de taak is om geruis te produceren.
Iedere oscillator lijkt voor de programmeur op een randapparaat, waarheen hij instelgegevens betreffende de frequentie (toon) en het volume (luidheid) kan verzenden. Ook het volume van de ruisgenerator kan op deze wijze worden ingesteld. Het bereik waarin de frequenties instelbaar zijn, is van 3Ø Hz tot 65 kHz. Dit ligt ver buiten de gehoorgrens. Deze loopt ongeveer van 4Ø Hz tot 18 kHz. Het geluid van de verschillende oscillatoren kan aan elkaar worden toe-

gevoegd. Het systeem zet dit geluid dan op het geluidskanaal van de TV en nog eens apart op een *connector*. Deze connector kan met een kabel aan een stereo-installatie worden verbonden voor de produktie van stereo-geluid. Op deze wijze is het mogelijk vele geluidseffecten te programmeren.
Wanneer we de oscillatoren nummeren van 1 tot en met 3, komt het stereogeluid als volgt tot stand:
Het geluid van oscillator 1 wordt toegevoegd aan het geluid van de ruisgenerator, daarbij wordt nog het geluid van oscillator 2 toegevoegd. Dit totaal wordt aan het linker stereo signaalkanaal afgegeven. Zo wordt de samenvoeging van de oscillatoren 3, 2 en de ruisgenerator als rechter stereokanaal uitgevoerd. De preciese details, die nodig zijn om in machinetaal de geluidsmogelijkheden te benutten, vindt u in de handleiding, die hoort bij de Teleac/DAI personal computer. We zullen er hier niet nader op ingaan. Wel zullen we net als bij het beeld zien hoe we de geluidsmogelijkheden op BASIC kunnen benutten.

18.5. Geluid met BASIC

Het eerste, dat we hier gebruiken, is het statement ENVELOPE. De bedoeling dat we van een geluid aangeven hoe groot het volume moet zijn en hoe lang het moet duren. Wat we eigenlijk doen is een soort 'noot' vastleggen. ENVELOPE zal steeds worden gebruikt als hulpfunctie bij het produceren van geluid.

Voorbeeld

```
ENVELOPE 1 1Ø,2Ø; 15,3Ø; 2,3 ; Ø
```

We stellen hiermee ENVELOPE nummer 1 als volgt samen: eerst een volume 1Ø gedurende 2Ø tijdseenheden, dan een volume van 15 gedurende 3Ø tijdseenheden en vervolgens een volume 2 gedurende 3 tijdseenheden, waarna we tenslotte het geluid afzetten.
De algemene vorm van dit statement is:

```
ENVELOPE NR V,L [; V,L] [; V,L]
```

NR	geeft het ENVELOPE NUMMER AAN
V	IS HET VOLUME
L	IS DE TIJDSDUUR

Het volume mag waarden van Ø (geluid is uitgezet) tot 15 (voluit geluid) hebben, terwijl er tijdsduren van 1 tot 254 tijdseenheden van ieder 3.4 milliseconde mogen worden gespecificeerd.
Er mogen maximaal 2 van deze ENVELOPES voor verder gebruik tegelijk worden klaargezet. Het begrip ENVELOPE stamt uit het idee dat de ENVELOPE als het ware een verpakking is voor het geluid, dat moet worden opgewekt.
Het bij ENVELOPE behorende statement is SOUND.

Daarbij moeten we het systeem vertellen welke oscillator moet worden gebruikt, welk ENVELOPE nummer hierbij hoort, welk volume de oscillator moet krijgen en ook welke frequentie.

Voorbeeld

```
SOUND 2 1 9 Ø FREQ (24ØØ)
```

Dit statement betekent dat oscillator 2 wordt gebruikt met EN-

VELOPE 1, dat de oscillator volume 9 krijgt en een frequentie gewenst is van 2400 Hz. De uitleg van de 0 (nul) in dit statement volgt later. Ten aanzien van het volume in SOUND en in ENVELOPE geldt dat ze voor het uiteindelijk ingestelde volume als met elkaar vermenigvuldigd moeten worden beschouwd.
De algemene vorm van het statement is:

SOUND OSC ENV VOL 0 FREQ

OSC	is het oscillatornummer	(1, 2 of 3)
ENV	is het ENVELOPE nummer	(0 of 1)
VOL	is het volume	(0 tot en met 15)
FREQ	is de frequentie	(31 tot en met 65 535 Hz)

Het getal 0 geeft aan, dat er geen *vibrato* of *glissando* moet worden gebruikt. Vibrato wil zeggen een gedefinieerde variatie in de toonsterkte (denk aan de opera).
Glissando heeft de betekenis van een gedefinieerde glijdende overgang van de ene toon naar de andere, dus als het ware wegstervend en dan weer aanzwellend.
In plaats van het getal nul is het ook mogelijk de getallen 1, 2 of 3 te kiezen. De betekenis er van is:

0	geen vibrato en geen glissando
1	wel vibrato en geen glissando
2	geen vibrato en wel glissando
3	wel vibrato en wel glissando

Tenslotte bestaat er nog de mogelijkheid om geruis te produceren met het statement NOISE. Daartoe moet weer de gewenste ENVELOPE opgegeven worden en het volume.

Voorbeeld

NOISE 1 7

Dit statement produceert geruis met de ENVELOPE 1 en het volume 7.

Tenslotte kennen we nog de mogelijkheid het geluid onder programmabesturing af te zetten. Dit gebeurt door SOUND OFF. Daarna worden alle geluidsoscillatoren afgezet. Willen we alleen 1 oscillator afzetten dan geven we: SOUND 1 OFF met als effect dat oscillator 1 wordt afgezet. Tenslotte kunnen we ook de ruis afzetten: NOISE OFF.
Geluid laat zich op papier moeilijk weergeven. Vandaar dat voorbeelden achterwege zijn gelaten.

18.6. Geluid en beeld in andere computers

Grotere computers beschikken gewoonlijk niet over de mogelijkheid rechtstreeks beeld en geluid te produceren. Als terminalscherm wordt daar namelijk geen TV-toestel gebruikt. Voor personal computers treden op dit punt nogal wat verschillen op. Het is dan ook verstandig die verschillen voor een personal computer, die u wenst te kopen, grondig te bezien en op hun gebruikswaarde te schatten.
Veel personal computers hebben niet de uitgebreide grafische en geluidsmogelijkheden die hier voor de Teleac/DAI personal computer

zijn beschreven. We bezien nu nog enige mogelijkheden zoals die in andere personal computers bestaan. Hieruit blijkt al dat er op dit punt geen standaard in BASIC gedefinieerd is.

18.7. Beeld via vaste grafische characters

Soms zijn in de charactergenerator behalve de letters en cijfers ook specifieke grafische symbolen opgenomen: rechthoekjes, strepen, hartjes, enz. Deze worden als het ware opgenomen in de characterset buiten de ASCII-symbolen. Het hartje kan bijvoorbeeld de waarde 173 in de character generator krijgen. Door middel van het BASIC statement PRINT CHR$ (173) roept men dit hartje op. De ASCII-tabel is daarbij dus uitgebreid tot 256 characters (normaal 128 characters). Van ieder character is de afbeelding vastgelegd.
Het plaatje wordt nu opgebouwd alsof gewone tekst wordt afgedrukt. Door op een handige wijze de af te drukken characters te kiezen, bouwt men een plaatje op. Een horizontale lijn ontstaat zo, door op een regel een serie characters af te laten drukken, die ieder een streepje weergeven. Aangezien de charactergenerator in principe aaneensluitende characters geeft, ontstaan dus geen gaten in de lijn. Voor de afstand tussen gewone letters en cijfers zijn namelijk in horizontale en verticale richting in de te genereren characters blanco stukjes geprogrammeerd. Deze blanco stukjes worden door de speciale grafische symbolen benut.

18.8. Beeld via instelbare grafische characters

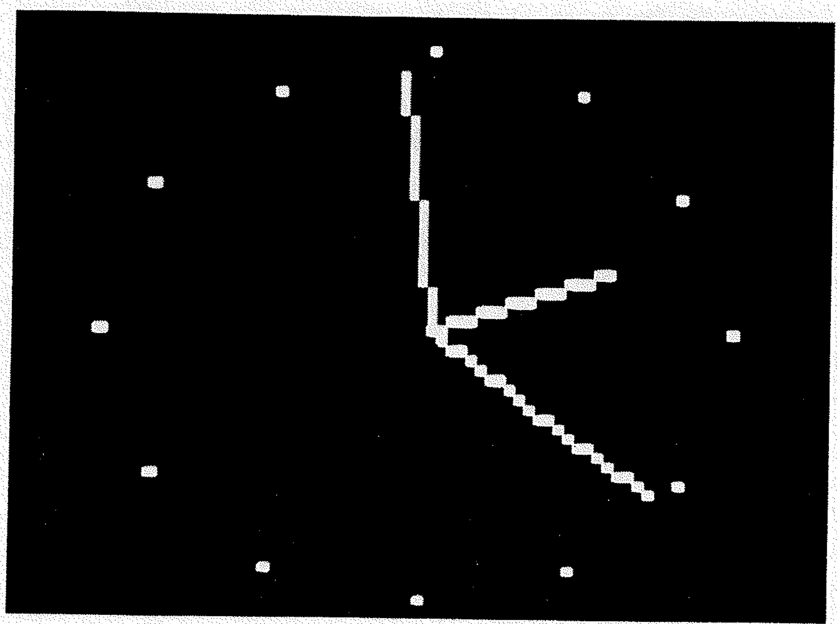
In het bovenstaande werd ervan uitgegaan dat de grafische characters voorgeprogrammeerd aanwezig waren. Daarvoor wordt door het systeem een (P)ROM in de charactergenerator benut. Wanneer we echter een deel van deze (P)ROM door RAM geheugen vervangen, kunnen we ook zelf characters definiëren. We schrijven dan de door ons verkozen bit-combinaties in het RAM-geheugen van de charactergenerator. We krijgen hier wat wel genoemd wordt een programmeerbare grafische characterset. Gewoonlijk zijn de gewone ASCII-tekens in (P)ROM aanwezig (128 stuks) en staan nog eens 128 tekens in RAM ter beschikking. De documentaire van de computer moet wel precies aangeven hoe we dit RAM gebiedje moeten vullen met de door ons beoogde characters.
Het voordeel van deze opzet is, dat we nu geheel zelf onze grafische characters kunnen bepalen. Op ieder moment kunnen we zelf die characters vervangen door anderen, doordat we ze gewoon overschrijven. Zo ontstaan uitgebreide mogelijkheden. We zijn daarbij nog wel steeds gebonden aan het feit dat we bij de opbouw van het plaatje uit moeten blijven gaan van een indeling in characters en niet van een indeling in beeldpunten.
We kunnen nu ook eenvoudig speciale characters zoals Griekse characters of afwijkende andere buitenlandse tekens in de tekst opnemen. Hierbij geldt nog steeds de waarschuwing, dat we erop moeten letten dat een aangekoppelde printer deze characters moet kunnen printen.
De Exidy Sorcerer computer hanteert de methode van instelbare grafische karakters. We kunnen daar dus willekeurige karakters in een tekst opnemen. Het is daarentegen moeilijk een willekeurig

plaatje uit beeldpunten op te bouwen. Om dit te doen moet de karakterset steeds veranderen en zelfs dan zijn er nog beperkingen. Uiteraard is het mogelijk dat we dit door een apart programma laten verzorgen. Zo'n programma, dat voor de Exidy te koop is, geeft ons dan toch weer vrijwel onbeperkte grafische mogelijkheden voor het maken van plaatjes binnen de beeldruimte die ter beschikking staan.

SPRAAK
Sommige personal computers voorzien zelfs in de mogelijkheid van spraak-uitvoer. U hoort dan als het ware een stem. Soms wordt de stem daarbij uit samengevoegde synthetische tonen gemaakt. In andere gevallen maakt men gebruik van een 'ingeblikte' menselijke stem. Zie weer uw computerhandleiding. Een enkele computer kan zelfs een beperkte hoeveelheid woorden herkennen als spraakinvoer. Geluid, spraak en plaatjes kunnen een wezenlijke factor betekenen in de communicatie tussen mens en computer. Op dit punt kunnen we dan ook nog vele ontwikkelingen verwachten. Vandaar dat er zeker nog geen standaard is aan te geven. De waarde hiervan voor uw eigen toepassing moet u zelf afwegen.

18.9. Voorbeeld

Tenslotte voegen we nog een aardig voorbeeld van het gebruik van de grafische mogelijkheden in de Teleac/DAI personal computer bij, namelijk een programma dat een klok op het beeldscherm weergeeft. Deze klok bevat een grote, een kleine en de secondewijzer. De precieze timing is proefondervindelijk vastgesteld door na te gaan hoeveel berekeningstijd nodig is om het beeld te maken iedere keer nadat met WAIT TIME op de juiste tijd gewacht is. In de praktijk bleek de klok goed gelijk te lopen. Uw eigen systeem zou iets langzamer of sneller kunnen zijn. Dan moet u de timing nog iets bijstellen.



```
*LIST
10    REM ******************
20    REM **  K  L  O  K  **
30    REM ******************
40    MODE 2A
45    REM ** INITIEER STAPWAARDE VAN DE WIJZERS IN RADIALEN **
50    S9=PI/30.0
60    M9=PI/1800.0
70    U9=PI/21600.0
80    INPUT "MINUTEN,UREN";M,U
90    MODE 2
100   REM ** TEKEN 12 PUNTEN **
110   FOR X=0.0 TO 2.0*PI STEP PI/6.0
120   DOT 30+30*SIN(X),30+30*COS(X) 15
130   NEXT X
140   REM ** BEREKEN STARTWAARDE VAN DE KLOK **
150   M3=M*PI/30.0
160   U3=U*PI/6.0+M3/12.0
170   S1=30.0:S2=30.0:M1=30.0:M2=30.0:U1=30.0:U2=30.0
180   REM ** TEKEN ELKE SECONDE DE WIJZERS **
190   FOR S3=0.0 TO 2.0*PI STEP S9
200   DRAW 30,30 S5,S6 0
210   DRAW 30,30 S1,S2 10
220   DRAW 30,30 M5,M6 0
230   DRAW 30,30 M1,M2 15
240   DRAW 30,30 U5,U6 0
250   DRAW 30,30 U1,U2 15
260   REM ** WACHT **
270   WAIT TIME 27
280   REM ** BEWAAR OUDE WIJZER COORDINATEN **
290   S5=S1:S6=S2
300   M5=M1:M6=M2
310   U5=U1:U6=U2
320   REM ** BEREKEN NIEUWE WIJZER COORDINATEN **
330   S1=30.0+28.0*SIN(S3)
340   S2=30.0+28.0*COS(S3)
350   M3=M3+M9
360   M1=30.0+28.0*SIN(M3)
370   M2=30.0+28.0*COS(M3)
380   U3=U3+U9
390   U1=30.0+20.0*SIN(U3)
400   U2=30.0+20.0*COS(U3)
410   NEXT S3
420   REM ** EEN MINUUT GEHAD **
430   GOTO 190
*
```

Samenvatting hoofdstuk 18

In dit hoofdstuk is getoond hoe het beeldscherm van de TV gebruikt kan worden om tekst en grafische informatie in zwart/wit en kleur zichtbaar te maken. Dit kan zowel onder besturing van BASIC als onder besturing van een programma dat in machinetaal is geschreven. Op dezelfde wijze is ook de geluidsprogrammering behandeld.

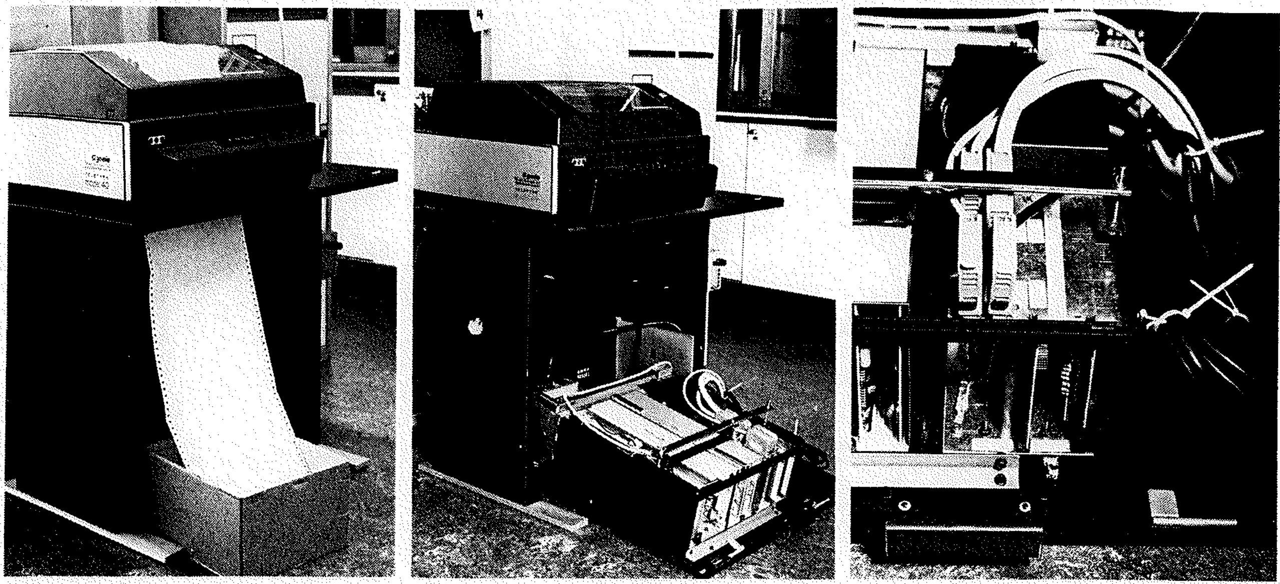
Hoofdstuk 19

I/O uitbreiding

In dit hoofdstuk gaan we nader in op de aspecten van I/O-uitbreidingen. We gaan zelf programma's in machinetaal schrijven, die een aan te sluiten randapparaat bedienen. Dit wordt met een voorbeeld van een printer geïllustreerd. We kijken naar de mogelijkheid van uitbreiding met een printer, een cassetterecorder en een modem, als die soms wenselijk is. Tenslotte gaan we in op de uitbreiding met een floppy disk tot een volwaardige computer voor bijvoorbeeld administratieve toepassingen.

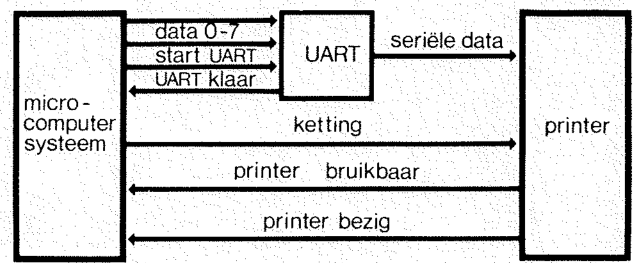
19.1. Programma's in machinetaal

Ondanks de reeds in hoofdstuk 17 opgesomde mogelijkheden in BASIC is het soms noodzakelijk of wenselijk een speciaal programma, dat rechtstreeks met de buitenwereld contact heeft, in machinetaal te schrijven. De besturing van de meeste randapparaten kan daarmee vaak het best geprogrammeerd worden, vooral omdat zo'n programma dan eenvoudig in PROM kan worden gezet. Het is dan natuurlijk noodzakelijk dat we zo'n programma in BASIC kunnen aanroepen. Nadat het programma is uitgevoerd, hervatten we het BASIC programma. Het programmeren in machinetaal zal in het algemeen sneller zijn (wat executiesnelheid betreft) dan een zelfde BASIC programma en ook minder geheugenruimte innemen.
In hoofdstuk 12 hebben we reeds gezien hoe een BASIC programma in machinetaal programma's kan aanroepen. De overdracht van gegevens tussen het BASIC programma en het programma in machinetaal verloopt via afgesproken geheugenplaatsen.
We zullen laten zien hoe een aangepaste printer, die geschikt is voor seriecommunicatie, in principe kan worden aangesloten. Als voorbeeld hiervan een printer van het fabrikaat Teletype model 40 (zie afbeelding 19.1).
Het gekozen voorbeeld is ontleend aan een situatie, waarin de printer ook werkelijk op deze wijze in gebruik is. We gaan ervan uit dat we de characters, die de printer moet afdrukken, in serie aanbieden zoals in hoofdstuk 14 is behandeld. Een UART (universal asynchrounus receiver and transmitter) zet de hem aangeboden *parallelle informatie* over in *seriële informatie.* (Seriële informatie kan ook weer teruggezet worden tot parallelle informatie.) We moeten de UART eerst 1 byte data aanbieden en vervolgens het signaal START UART geven. Op grond van dit signaal begint de UART het byte om te zetten in serie-informatie. Als de omzetting klaar is, bieden we opnieuw 1 byte aan en geven weer het signaal START UART.



Afb. 19.1. Regelprinter Teletype model 40

De serie-uitgang van de UART wordt met een kabel naar de ingang van de printer gevoerd. Om te weten of de UART klaar is, moeten we het signaal UART KLAAR bezien, dat de UART afgeeft als de omzetting klaar is.
In de vorige hoofdstukken hebben we de USART behandeld. Deze kan ook nog synchrone overdracht aan. Vandaar de S in de naam. In dit hoofdstuk gebruiken we alleen asynchrone overdracht. Een vereenvoudigde versie van de USART, nl. de UART is dan voldoende.
De signalen tussen het microcomputersysteem, de UART en de printer worden nog eens weergegeven in afbeelding 19.2.



Afb. 19.2. Aansluiting printer

Het model 40 is een *kettingprinter*, dat wil zeggen dat de characters op een ronddraaiende ketting aanwezig zijn. Met een hamertje wordt steeds een klap tegen het gewenste character gegeven. Daardoor slaat het character met het inktlint tegen het papier en drukt het desbetreffende teken af. Dit kan gebeuren met een maximale snelheid van 120 characters per seconde. We zijn in staat de ketting te laten lopen of te stoppen. Het lopen van de ketting maakt uiteraard lawaai, daarom is het gewenst de ketting te stoppen als er niet wordt geprint. Daarvoor dient het signaal KETTING. Als de ketting moet lopen moet dit signaal hoog gemaakt worden.

Van de printer ontvangen we de volgende signalen:

LINEPRINTER BRUIKBAAR

Een signaal dat aangeeft dat de printer kan gaan printen. Het impliceert dat de voeding aan staat, dat er nog papier en lint aanwezig is, terwijl de kast goed dicht zit.

PRINTER BEZIG

De printer kan nu geen gegevens ontvangen, omdat hij nog met printen bezig is of omdat hij bezig is het papier omhoog te schuiven voor een nieuw blad. Wanneer we nu geen gegevens blijven overzenden kan de printer deze niet afdrukken, ze gaan dus verloren.

De signalen tussen de computer en de buitenwereld zijn steeds laag actief gekozen, wat wil zeggen dat het signaal Ø is als het naar onze wens optreedt. We starten de UART als het signaal START UART laag (Ø) is. Dit geldt ook voor de ketting enz. Dit is om de volgende redenen gedaan: de *aanpassing* (dat is de schakeling tussen het randapparaat en de computer) is bij de printer ingebouwd. Er loopt dus een kabel van de computer naar de printer. Wanneer in deze kabel nu een draad breekt (bijvoorbeeld START UART), lijkt het aan de printerzijde alsof het signaal hoog (1) is. Dit wordt veroorzaakt door de elektrische eigenschappen van de componenten in de aanpassing. Ten gevolge van de kabelbreuk zal de printer dus nooit zo maar gaan printen, want voor het printcommando zouden de binnenkomende signalen laag (Ø) moeten zijn.

We gaan hier niet in op de preciese componenten die in de aanpassing aanwezig zijn. Dat zou ons wat dit boek betreft te ver in details van elektronische componenten voeren. De nadruk ligt hier namelijk op het programmeren. Vandaar dat we wel uitgebreid ingaan op het programma, dat de printer aanstuurt.

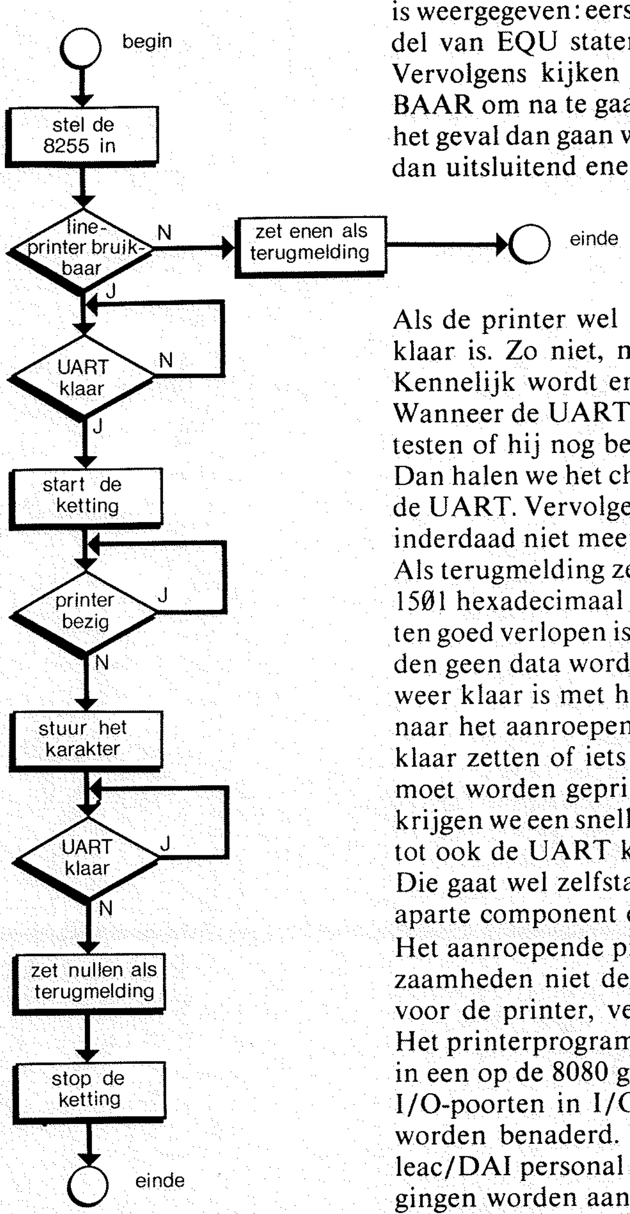
```
0001               ADRES     EQU   1      ;ADRES VAN DE OUTPUTPOORT
0000                         ORG   1200H  ;BEGINADRES VAN HET PROGRAMMA
1200    3E01       BEGIN:    MVI   A,01H  ;ZET 1 IN DE ACCU
1202    D301       WEER:     OUT   ADRES  ;STUUR ACCUWAARDE NAAR DE OUTPUTPOORT
1204    07                   RLC          ;VERSCHUIF ACCUWAARDE 1 PLAATS NAAR
                                          ;LINKS.
1205    C30212               JMP   WEER   ;OPNIEUW SCHUIVEN EN UITVOEREN
1200                         END   BEGIN

NO PROGRAM ERROR(S)
```

```
                 ;***********************************************************************
                 ;
                 ;
                 ;       LINEPRINTERPROGRAMMA
                 ;       VOOR
                 ;       TELETYPE MODEL 40
                 ;       HET TE PRINTEN CHARACTER STAAT OP ADRES 1500H
                 ;
                 ;       ALS HET PRINTEN SUCCESVOL VERLOPEN IS WORDT ADRES
                 ;       1501H GEVULD MET NULLEN.
                 ;       ZONIET,DAN MET ENEN.
                 ;***********************************************************************
                 ;
                 ;
0000                     ORG     1502H     ;STARTADRES VAN HET PRINTERPROGRAMMA
0001             OPDAT   EQU     1         ;DATA OUTPUTPOORT
0002             OPCOM   EQU     2         ;COMMANDO OUTPUT POORT
0000             IPSTA   EQU     0         ;STATUS INPUT POORT
0002             LBRUIK  EQU     02H       ;LINEPRINTER BRUIKBAAR BIT
0010             UKLAAR  EQU     10H       ;UART KLAAR BIT
0004             PBEZIG  EQU     04H       ;PRINTER BEZIG BIT
1500             KADRES  EQU     1500H     ;ADRES WAAR HET CHARACTER STAAT
1501             TADRES  EQU     1501H     ;ADRES VOOR DE TERUGMELDING
0010             SUART   EQU     10H       ;START UART BIT
0003             KETTIN  EQU     03H       ;KETTING AANZETTEN
00FF             ENEN    EQU     0FFH      ;ALLEMAAL ENEN
0000             NULLEN  EQU     0         ;ALLEMAAL NULLEN
                 ;
                 ;
                 ;***********************************************************************
                 ;
                 ;       TEST OF DE PRINTER BRUIKBAAR IS
                 ;
                 ;***********************************************************************
1502   DB00      LPDRV:  IN      IPSTA     ;LEES LINEPRINTER STATUS
1504   2F                CMA               ;INVERTEER VOOR POSITIEVE LOGICA
1505   F602              ANI     LBRUIK
1507   CA1015            JZ      LPDR1     ;O.K. PRINTER IS BRUIKBAAR
150A   3EFF              MVI     A,ENEN    ;ACCU WORDT 11111111
150C   320115            STA     TADRES    ;ZET TERUGMELDING:FOUT
150F   C9                RET               ;GA TERUG NAAR AANROEPEND PROGRAMMA
```

```
                        ;
                        ;**************************************************************
                        ;
                        ;       TEST OF DE UART KLAAR IS
                        ;
                        ;**************************************************************
                        ;
1510    DB00            LPDR1:  IN      IPSTA     ;LEES DE STATUS
1512    2F                      CMA               ;INVERTEER
1513    E610                    ANI     UKLAAR
1515    CA1015                  JZ      LPDR1     ;WACHT TOT UART KLAAR IS
1518    3F03                    MVI     A,KETTIN  ;START DE KETTING
151A    2F                      CMA               ;INVERTEER
151B    D302                    OUT     OPCOM     ;NU ECHT STARTEN
                        ;
                        ;**************************************************************
                        ;
                        ;       VERZEND HET CHARACTER
                        ;
                        ;**************************************************************
                        ;
151D    DB00            LPDR2:  IN      IPSTA     ;LEES DE STATUS
151F    2F                      CMA               ;INVERTEER
1520    E604                    ANI     PBEZIG    ;BEZIG?
1522    C21D15                  JNZ     LPDR2     ;JA:DAN WACHTEN
1525    3AC015                  LDA     KADRES    ;HAAL HET CHARACTER OP
1528    2F                      CMA               ;INVERTEER
1529    D301                    OUT     OPDAT     ;EN ZET CHARACTER OP DE DATA LIJNEN
152B    3E13                    MVI     A,SUART OR KETTIN  ;START DE UART EN DE KETTING
152D    2F                      CMA               ;INVERTEER
152F    D302                    OUT OPCOM         ;NU STARTEN
                        ;
                        ;**************************************************************
                        ;
                        ;       TEST OF DE UART MET VERSTUREN IS BEGONNEN
                        ;
                        ;**************************************************************
                        ;
1530    DB00            LPDR4:  IN      IPSTA     ;LEES STATUS
1532    2F                      CMA               ;INVERTEER
1533    E610                    ANI     UKLAAR    ;KIJK OF DE UART KLAAR IS
1535    C23015                  JNZ     LPDR4     ;WACHT TOT DE UART KLAAR IS
1538    3E00                    MVI     A,NULLEN  ;ZET NULLEN ALS TERUGMELDING
153A    320115                  STA     TADRES    ;OP HET JUISTE ADRES.
153D    2F                      CMA               ;INVERTEER
153E    D302                    OUT     OPCOM     ;ZET PRINTER WEER AF
1540    C9                      RET               ;KEER TERUG NAAR HET AANROEPEND PROGRAMMA
0000                            END

NO PROGRAM ERROR(S)
```



Afb. 19.3. Flowchart TTY model met besturing

Wanneer we naar het printer-programma kijken, zien we daar de volgende onderdelen zoals ook in de flowchart van afbeelding 19.3 is weergegeven: eerst een aantal toekenningen van namen door middel van EQU statements om het programma leesbaar te houden. Vervolgens kijken we naar het signaal LINEPRINTER BRUIKBAAR om na te gaan of de printer kan worden gebruikt. Is dat niet het geval dan gaan we uit de routine met een foutmelding. We zetten dan uitsluitend enen op geheugenplaats 15Ø1H.

Als de printer wel kan worden gebruikt, gaan we na of de UART klaar is. Zo niet, moeten we wachten totdat dit wel het geval is. Kennelijk wordt er door de UART nog een character verzonden. Wanneer de UART klaar is, starten we de ketting van de printer en testen of hij nog bezig is met het printen.
Dan halen we het character op, zetten het op de datalijnen en starten de UART. Vervolgens wachten we weer en controleren of de UART inderdaad niet meer ready is en dus met de verzending is begonnen. Als terugmelding zetten we nu uitsluitend nullen in geheugenplaats 15Ø1 hexadecimaal (ofwel 5377 decimaal). Dit geeft aan dat het printen goed verlopen is. De ketting stopt nu automatisch als enige seconden geen data wordt aangeboden. We wachten niet totdat de UART weer klaar is met het verzenden van de gegevens, maar gaan direct naar het aanroepende programma terug. Dat kan dan nieuwe data klaar zetten of iets geheel anders gaan doen. Wanneer er opnieuw moet worden geprint, testen we eerst wel of de UART klaar is. Zo krijgen we een snellere verwerking dan wanneer we meteen wachten tot ook de UART klaar is.
Die gaat wel zelfstandig door als hij gestart is. Het is namelijk een aparte component die zijn opdracht zal afwerken.
Het aanroepende programma mag bij het voortgaan van zijn werkzaamheden niet de inhoud van de I/O-poorten, die wij gebruiken voor de printer, veranderen, want dat zou het printen verstoren. Het printerprogramma zoals dat hier is weergegeven, is geschikt om in een op de 8080 gebaseerde microcomputer te werken, waarbij de I/O-poorten in I/O mapped en niet in memory-mapped methode worden benaderd. Voor de memory-mapped-I/O, zoals in de Teleac/DAI personal computer is gebruikt, moeten de volgende wijzigingen worden aangebracht.
Ten eerste worden de IN en OUT instructies vervangen door bijvoorbeeld:

OUT	MODPRT	wordt:	LXI	H,MODPRT
			MOV	M,A
IN	IPSTA	wordt:	LXI	H,IPSTA
			MOV	A,M
OUT	OPCOM	wordt:	LXI	H,OPCOM
			MOV	M,A
OUT	OPDAT	wordt:	LXI	H,OPDAT
			MOV	M,A

uiteraard moeten ook de waarden voor MODPRT, IPSTA, OPCOM, OPDAT worden aangepast tot een geheugenadres van 16 bits.

MODPRT EQU 3	wordt:	MODPRT EQU	FEØ3H
IPSTA EQU Ø	wordt:	IPSTA EQU	FEØØH
OPCOM EQU 2	wordt:	OPCOM EQU	FEØ2H
OPDAT EQU 1	wordt:	OPDAT EQU	FEØ1H

Een voorbeeldprogramma in machinetaal dat het printerprogramma aanroept om tekst te laten printen is weergegeven als programma 19.2.

```
                   ;************************************************************
                   ;
                   ;       AANROEPEN VAN DE PRINTER
                   ;
                   ;************************************************************
                   ;
0090               MODE    EQU   90H       ;CODE VOOR DE 8255
0003               MODPRT  EQU   3         ;MODEPOORT VOOR DE 8255
00FF               ENEN    EQU   0FFH      ;ALLEMAAL ENEN
0001               OPDAT   EQU   1         ;PRINTER DATA POORT
0002               OPCOM   EQU   2         ;PRINTER COMMANDO POORT
1500               KADRES  EQU   1500H     ;ADRES WAAR HET CHARACTER WORDT GEZET
1502               LPDRV   EQU   1502H     ;ADRES VAN DE PRINTER ROUTINE
                   ;
1200  310012       START:  LXI   SP,1200H  ;INITIEER DE STACK POINTER
1203  3E90                 MVI   A,MODE    ;ZET 8255 MODE
1205  D303                 OUT   MODPRT    ;IN 8255 MODE REGISTER
1207  3EFF                 MVI   A,ENEN    ;MAAK DE UITGANGEN HOOG
1209  D301                 OUT   OPDAT     ;OM DE PRINTER UIT TE ZETTEN
120B  D302                 OUT   OPCOM
120D  211F12               LXI   H,STRING  ;
1210  0632                 MVI   B,AANTAL  ;AANTAL LETTERS IN B ZETTEN
1212  7E           LOOP:   MOV   A,M       ;CHARACTER IN DE ACCU
1213  320015               STA   KADRES    ;ZET CHARACTER OP 1500H
1216  CD0215               CALL  LPDRV     ;PRINT NU HET CHARACTER !!!!!!!
1219  23                   INX   H         ;STAP NAAR HET VOLGENDE CHARACTER
121A  05                   DCR   B         ;TEST OP AANTAL RESTERENDE CHARACTERS
121B  C21212               JNZ   LOOP      ;ALS ER NOG OVER ZIJN: PRINT OPNIEUW
121E                       RST   1         ;GA NAAR SYSTEEMMONITOR
                   ;
121F  44697420     STRING: DB    'Dit is een zinnetje om de teleacdriver '
1223  69732065
1227  656E207A
122B  696E6E65
122F  746A6520
1233  6F6D2064
1237  65207465
123B  6C656163
123F  64726976
1243  657220
1246  74652074         DB    'te testen',0DH,0AH
124A  65737465
124E  6E0D0A
                  EINDE:
0032              AANTAL  EQU   EINDE-STRING
1200                      END   START

NO PROGRAM ERROR(S)
```

We beginnen daar uiteraard de stackpointer een waarde te geven om te voorkomen dat deze straks bij de subroutine-aanroep van het printer-programma op een willekeurige geheugenplaats wijst. Als het printerprogramma door andere programma's wordt aangeroepen, moet dit programma de stackpointer zetten. De stackpointer zou dan ongecontroleerd naar bijvoorbeeld ons programma kunnen wijzen. Aangezien bij de subroutine-aanroep in de stack geschreven wordt, kan door een foutieve stackpointer ons programma worden vernield. Vandaar: eerst de stack initialiseren. Daarna zetten we de 8255 in de goede mode. Vervolgens zetten we enen op de uitgang naar de printer om te voorkomen dat deze zo maar begint (laag-actieve signalen). Dan laden we HL met het beginadres van de te printen string. B wordt geladen met de lengte van de string AANTAL. Opvallend daarbij is, dat we AANTAL berekenen uit het eindadres van de string minus het beginadres. Dan gaan we door in de loop. Daar halen we steeds 1 character op uit het geheugen. We zetten dat op het adres, waar de printerroutine het verwacht (KADRES). We roepen het printerprogramma aan (subroutine call). We wijzen het volgend character in het geheugen aan. Tenslotte verlagen we het aantal nog te printen characters met 1. Wanneer de telling nog niet nul is, printen we opnieuw. Is de telling wel nul, dan zijn we klaar.
Er zijn 2 opvallende aspecten:
We testen niet of het printen wel of niet goed verlopen is. Het is een leuke oefening om dat er bij te programmeren. Deze test is hier achterwege gelaten. Het zou beter geweest zijn, wanneer we de instelling van de 8255 in het printerprogramma hadden uitgevoerd. Voordat we een character printen stellen we dan de 8255 in de goede mode in. Echter bij iedere mode-instelling worden alle uitgangen gereset (op nul gezet). Dus dan zou de UART meteen gestart worden, voordat we de goede data aangeboden hadden (laag actieve lijnen). Resultaat: chaos op de printer. Dit probleem zou oplosbaar zijn met de keuze van hoog actieve signalen op de kabel. Dat is echter om al eerder genoemde betrouwbaarheidsredenen onwenselijk. Inverteren (omdraaien) van de signalen in de aanpassing en aan de computerzijde ware ook mogelijk. Maar we wensten de apparatuur voor dit voorbeeld niet te veranderen, omdat hij ook in andere opstellingen wordt gebruikt. Ergo: kiezen tussen 2 kwaden, deed ons besluiten de software opbouw iets minder fraai te maken. Bovendien geeft dit een aardige illustratie van problemen zoals die zich in de praktijk nog al eens zullen voordoen. Aan het eind van dit boek mag u dus ook het niet-ideale geval wel eens zien, in afwijking van alle andere gegeven ideale voorbeelden.
Het voorbeeldprogramma in machinecode dat uitprint 'Dit is een zinnetje om de Teleac driver te testen' zou fout lopen in de samenwerking met het voor memory-mapped-I/O aangepaste printerprogramma. De registers H en L worden namelijk in die beide programma's gebruikt om naar geheugenplaatsen te verwijzen. In het printerprogramma worden daar in feite I/O-poorten geadresseerd. Het is een nuttige oefening deze beide memory-mapped programma's zo te schrijven, dat dit tweeledig gebruik van H en L in goede banen wordt geleid.
Een programma dat in BASIC het printerprogramma aanroept, zou kunnen zijn:

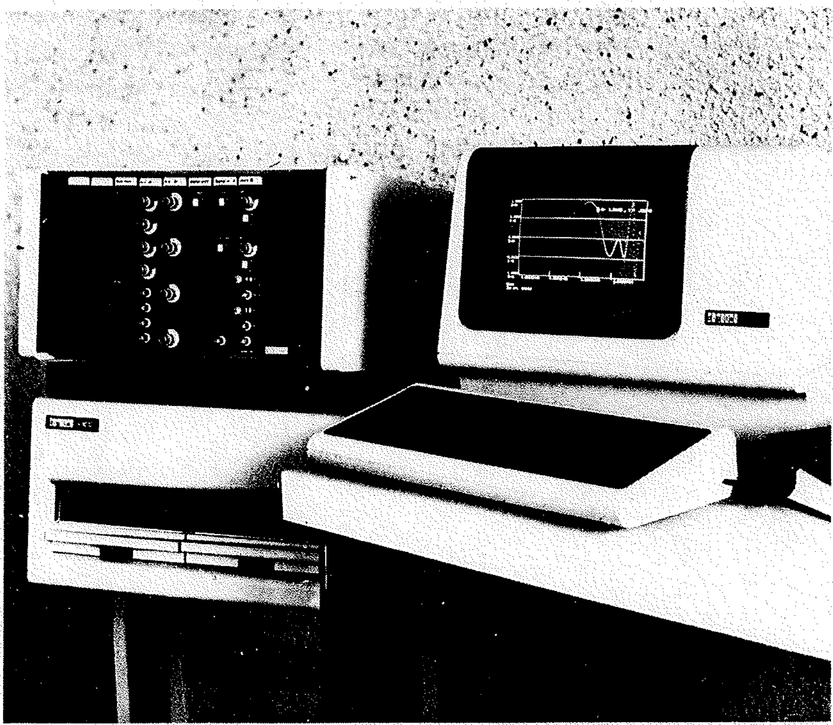
```
*
*LIST
10    REM  LAAT EEN ZIN OP DE PRINTER AFDRUKKEN
20    LET ZIN$="Dit is een zin om de teleac driver te testen"
30    REM  VOEG EEN CARRAIGE RETURN EN EEN LINE FEED TOE
40    LET ZIN$=ZIN$+CHR$(13)+CHR$(10)
50    FOR J=0.0 TO LEN(ZIN$)-1.0
60    LET KARAKTER=ASC(MID$(ZIN$,J,1))
65    PRINT KARAKTER
70    POKE #1500,KARAKTER
80    CALLM #1502
90    IF PEEK(#1501)=0 GOTO 130
100   PRINT "MAAK DE PRINTER OPERATIONEEL"
110   PRINT "EN GEEF DAN: CONTINUE"
120   STOP
130   NEXT J
140   PRINT "KLAAR MET PRINTEN"
150   END
*
```

We gebruiken daarin weer de zin 'Dit is een zin om de Teleac driver te testen'. Om nog eens te herinneren aan het feit dat we met CHR$ ASCII-characters kunnen produceren, voegen we in regel 40 een *carriage return* en een *linefeed* toe om na het printen weer op een nieuwe regel te beginnen.
Regel 60 haalt het te printen character (ofwel de letter) uit de zin. Regel 70 zet dit character op de afgesproken adresplaats 1500 hexadecimaal (of 5376 decimaal). Dan roepen we het printerprogramma aan via regel 80. Regel 90 controleert of het printen goed is verlopen. Zo ja, dan printen we het volgende character. Zo nee, dan melden we dat de printer niet operationeel is: regels 100 en 110. We stoppen en wachten op CONTINUE om verder te gaan zoals regel 120 weergeeft. U weet nog: na STOP moeten we CONTINUE geven om verder te kunnen gaan. We handelen via regels 50 tot en met 130 het printen van alle characters van de zin af.
Om de geprinte characters ook op ons scherm zichtbaar te maken voegen we regel 65 toe.

19.2. Printer

We hebben reeds gezien hoe in principe een bepaalde printer op een microcomputersysteem kan worden aangesloten. Nu is dit probleem natuurlijk niet nieuw. Op veel computers zijn reeds printers aangesloten. Om te voorkomen dat daarbij steeds opnieuw aanpassingen in apparatuur en programmatuur moeten worden gemaakt is er een standaard vastgelegd voor zo'n printeraansluiting. Deze standaard wordt aangeduid met RS 232-C of ook wel met V24.
De standaard legt dus precies vast hoe een computer met een printer kan worden verbonden. Omdat er allerlei soorten printers en computers zijn, beslaat de standaard verschillende mogelijkheden. We geven hier uitsluitend een korte beschrijving voor de aansluiting van een terminal. Deze gedraagt zich in feite als een printer voor de uitvoer en als toetsenbord voor de invoer. Toetsenbord en printer

zijn niet mechanisch gekoppeld zoals bij een schrijfmachine. Een ingetypt character wordt van het toetsenbord naar de computer verzonden. Deze zendt op zijn beurt het character naar de printer. Zo kan de gebruiker meteen zien of zijn ingetypte character bij de computer is gearriveerd. Deze methode staat bekend onder de naam FULL DUPLEX methode.
De characters zelf worden via een serieverbinding naar de computer verzonden. Zo wordt het character via een andere serieverbinding naar de printer gestuurd. Deze signaallijnen worden aangeduid met de namen RECEIVE DATA en TRANSMIT DATA.
Nota Bene: bij de naamgeving wordt steeds vanuit de computer gedacht.
Het is belangrijk op te merken dat zowel de computer als de terminal klaar zijn om te kunnen werken. Daarvoor dienen de signalen DATA SET READY (computer is klaar) en DATA TERMINAL READY (terminal is klaar).
Tenslotte wordt de overdracht van de gegevens begeleid door 2 signalen om aan te geven dat er gegevens klaar staan en verzonden mogen worden. De computer verstuurt dan een signaal CLEAR TO SEND (klaar om te zenden). De terminal verzendt REQUEST TO SEND (verzoek om te zenden).
Via deze 6 signalen is de overgrote meerderheid van terminals printers en dergelijke aan te sluiten. Uiteraard hoort hierbij nog de aardleiding, GROUND geheten. De signaallijnen sturen hun signalen als spanningen van plus en min 12 volt over de lijnen.
De RS 232-C standaard beschrijft in totaal 25 signalen. De plaats van ieder signaal in een connector en de vorm en de maten van de connector met 25 pennen zijn precies vastgelegd. Gewoonlijk kan men met de 6 genoemde signalen volstaan (zie afbeelding 19.4).



MINC is een computersysteem voor laboratorium-toepassingen. Het systeem bestaat uit een in de beeldscherm-terminal ingebouwde centrale verwerkingseenheid, een dubbele floppy-disk en 8 speciale functie-modules, die rechtstreeks met laboratorium-apparatuur verbonden kunnen worden. MINC is programmeerbaar in BASIC

Pin Number	Circuit	Description
1	AA	Protective Ground
2	BA	Transmitted Data
3	BB	Received Data
4	CA	Request to Send
5	CB	Clear to Send
6	CC	Data Set Ready
7	AB	Signal Ground (Common Return)
8	CF	Received Line Signal Detector
9	–	(Reserved for Data Set Testing)
10	–	(Reserved for Data Set Testing)
11		Unassigned (See section 3.2.3)
12	SCF	Sec. Rec'd. Line Sig. Detector
13	SCB	Sec. Clear to Send
14	SBA	Secondary Transmitted Data
15	DB	Transmission Signal Element Timing (DCE Source)
16	SBB	Secondary Received Data
17	DD	Receiver Signal Element Timing (DCE Source)
18		Unassigned
19	SCA	Secondary Request to Send
20	CD	Data Terminal Ready
21	CG	Signal Quality Detector
22	CE	Ring Indicator
23	CH/CI	Data Signal Rate Selector (DTE/DCE Source)
24	DA	Transmit Signal Element Timing (DTE Source)
25		Unassigned

Afb. 19.4. RS 232-C standaard

De kabellengte tussen computer en terminal mag maximaal 15 m zijn. De snelheid waarmee gegevens kunnen worden overgedragen, is maximaal 19 2ØØ bits per seconde. Gewoonlijk kan een terminal met papierafdrukeenheid *(printer terminal)* een maximale snelheid van 3ØØ bits per seconde aan. Wanneer in plaats van op papier op een scherm wordt afgedrukt, ligt deze snelheid gewoonlijk boven de 12ØØ bits per seconde. Een gewone printer haalt gewoonlijk maximaal 12ØØ bits per seconde.

Een character wordt vaak verzonden in de vorm van 1Ø bits. 1 bit wordt gebruikt als startbit, dan volgen 7 bits van het ASCII-karakter, het pariteitsbit en daarna 2 bits stopbits, zoals in hoofdstuk 14 reeds is behandeld.

De Teleac/DAT personal computer bezit aan de achterzijde een aansluiting volgens de RS 232-C standaard. Gegevens, die naar het beeldscherm worden gestuurd, gaan ook naar die aansluiting. Een printer, die wordt aangesloten volgens deze standaard, zal dus automatisch dezelfde gegevens printen als de gegevens die op het beeldscherm verschijnen. Op deze wijze hebben de auteurs de verschillende listings van programma's en afdrukken van de uitvoering van de programma's geproduceerd. De gebruiker hoeft dus

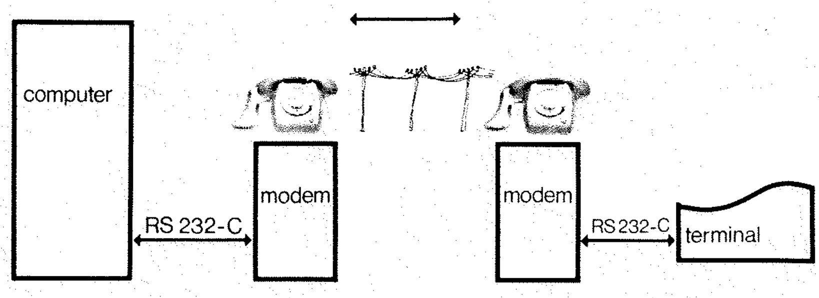
geen extra aanpassing in de vorm van apparatuur en programmatuur aan te brengen om een standaard RS 232-C printer aan te sturen. Omdat de toegestane 15 m kabel soms te weinig is gebruiken veel fabrikanten van computers en terminals nog een andere standaard, namelijk een zogenaamde current-loop (stroomlus).
Op de signaallijnen wordt dan niet met een spanning gewerkt, maar met een stroom. Deze bedraagt gewoonlijk 20 mA. Daarmee kan ongeveer 300 m kabel worden aangestuurd. Tevens wordt daarbij de definitie van RS 232-C nog vereenvoudigd. Er wordt domweg aangenomen dat de beide partners (computer en terminal) klaar zijn en in staat zijn om data te ontvangen. Kortom, signalen als DATA SET READY, DATA TERMINAL READY, CLEAR TO SEND en REQUEST TO SEND worden niet gebruikt. Men neemt domweg aan dat de verzending van de gegevens te allen tijde kan plaatsvinden. Er blijven dan slechts 4 draden over: RECEIVE DATA, TRANSMIT DATA en twee terugkerende (RETURN) leidingen. Ook deze manier van aansluiten van terminals is populair. Ze is bruikbaar wanneer de beide partners inderdaad kunnen garanderen steeds voor de verzending van gegevens beschikbaar te zijn.
Bij de meeste terminals en computers kan men kiezen tussen een aansluiting volgens RS 232-C of current-loop. Het komt daarbij vaak voor dat dan ook de RS 232-C aansluiting met 3 draden (RECEIVE DATA, TRANSMIT DATA, GROUND) werkt. Voor de software zijn beide methodes dan gelijk. Het verzendmechanisme neemt domweg aan dat beide partners klaar zijn.
Behalve printers, die op serie-basis werken, zoals hierboven is beschreven, bestaan er ook printers die op parallelle basis werken. In feite is er voor de programmeur weinig verschil. De data worden nu niet door de UART naar serie omgezet. Maar de programmeur had eigenlijk alleen maar te maken met parallelle data. Er zullen altijd nog signalen nodig zijn om het overzenden van de data te begeleiden. Deze signalen zijn weer vergelijkbaar met de bovengenoemde signalen CLEAR TO SEND en REQUEST TO SEND. Het is natuurlijk mogelijk daar nog een scala aan opdracht- en foutmeldingssignalen aan toe te voegen. Voor iedere printer moet dus worden bekeken hoe de aansluiting precies verloopt. Gelukkig leveren de meeste leveranciers van personal computers er printers bij. De aansluiting in hard- en software aan de computer is dan al volledig verzorgd. Als u weinig problemen wilt hebben, kiest u gewoon zo'n volledig verzorgde printer. Let er wel op dat goede printers ten opzichte van de computer zelf vrij duur zijn.

19.3. Modem

Wanneer de terminal zich op nog grotere afstand bevindt, kan men aan de computer een zogenaamd *modem* (MOdulator/DEModulator) koppelen. Dit modem zet de RECEIVE DATA en TRANSMIT DATA signalen om in geluidstonen. Deze geluidstonen kunnen dan via het telefoonnet worden verzonden want ze liggen in het gebied van de spraak.

Wanneer nu ook de terminal van een modem wordt voorzien, kan deze de geluidstonen weer omzetten in de gewenste digitale signalen en aan de terminal toevoeren. Zo is het mogelijk koppelingen tussen

computer en terminal over zeer grote afstanden te maken (zie afbeelding 19.5).
Voor het gewone telefoonnet moet men dan rekenen op een snelheid van 3ØØ bits per seconde (3Ø characters per seconde). Ook 12ØØ bits per seconde komt wel voor. In plaats van bits per seconde wordt ook wel de naam *baud* gebruikt, dus bijvoorbeeld 3ØØ baud. Men kan van de PTT evenwel aparte lijnen huren waarop snellere verzending mogelijk is (bijvoorbeeld 9600 baud).



Afb. 19.5. Gebruik van een modem

19.4. Toetsenbord en GETC

We hebben ons vooral geconcentreerd op de printer, dus het uitvoergedeelte. Vaak zal de printer gecombineerd zijn met een toetsenbord. Via de RS 232-C verbinding kunnen we de gegevens van het toetsenbord dan naar binnen voeren. De personal computer heeft zelf ook een toetsenbord. Het INPUT statement in BASIC vraagt nadrukkelijk gegevens van dat toetsenbord op. Het statement blijft wachten totdat de gebruiker iets ingetypt heeft, gevolgd door een CARRIAGE RETURN (terugloop-wagen). Intussen is de rest van het programma in feite werkeloos: er wordt op de gebruiker gewacht. Het is wenselijk over de mogelijkheid te beschikken om te kunnen kijken of de gebruiker iets heeft ingetypt, maar daarop niet te blijven wachten. Daarvoor kent BASIC veelal het statement GETC. Dit staat voor GET CHARACTER ofwel haal het character op.
Het effect van 1Ø A = GETC is dat de ASCII-waarde van het nu ingetypte character in A wordt geplaatst. Als er geen character zal zijn ingetypt, krijgt A de waarde Ø. Zo is dus na te gaan of er een character is ingetypt. GETC blijft niet op het typen wachten. Dit statement meldt slechts of er iets ingetypt is. Na uitvoering van GETC doet het systeem alsof er niets is ingetypt. Bij de volgende GETC zal dus alleen worden gemeld dat er iets is ingetypt, als dit inderdaad sinds de vorige GETC is gebeurd. Vooral in besturingstoepassingen kan GETC waardevolle gebruiksmogelijkheden opleveren.

19.5. Cassetterecorder

In plaats van hoorbare geluidstonen via een telefoonlijn te verzenden is het ook mogelijk deze geluidstonen op een gewone cassetterecorder op te nemen. Wanneer we dan met de cassetterecorder naar het modem van de printer gaan, zal deze de opgenomen gege-

vens uitprinten. Op dit principe zijn de aansluitingen voor cassetterecorders van personal computers gebaseerd. Men zet dan de seriële signalen van de RS 232-C aansluiting op eenvoudige wijze om in geluidssignalen.
Een veel gebruikte manier zet een bit met de waarde 1 om in 8 cycli van 2400 Hz (1 Hz is één trilling per seconde: 2400 Hz dus 2400 trillingen per seconde. Ons gehoor kan tonen waarnemen van ongeveer 40-18000 Hz). Een 0 wordt omgezet in 4 cycli van 1200 Hz. De tijd van 1 bit blijft dus steeds gelijk. De apparatuur die nodig is om deze omzetting heen en terug te verzorgen, is relatief eenvoudig. De programmatuur hiervan verschilt in wezen ook niet van die voor een terminal. Zo kan dus een cassetterecorder geschikt worden gemaakt voor de opslag van gegevens.
Wanneer men die gegevens nu nog van een naam voorziet, is het later eenvoudig na te gaan om welke gegevens het gaat. De naam wordt ook op de band geschreven.
Zo'n pakket van naam en gegevens wordt op het cassettebandje vooraf gegaan door een af te spreken geluidspatroon. Ook achteraf komt een bepaald patroon. Zo is het voor de computer na te gaan of er zich echt gegevens of zo maar chaos (muziek bijvoorbeeld) op de band bevindt.
De Teleac/DAI personal computer kan gegevens op een cassetteband opslaan en terugzoeken. Daarvoor zijn voor programma's de BASIC commando's (SAVE (zet weg) en LOAD (haal op) aanwezig. De gebruiker kan zijn programma's van een naam voorzien (maximaal 14 characters) bij SAVE. In LOAD kan ook weer een naam gespecificeerd worden. De computer zoekt dan alleen het programma op met die gespecificeerde naam. Wordt bij LOAD geen naam meegegeven, dan wordt domweg het eerstvolgende programma ingelezen. Dit gebeurt in de Teleac/DAI computer met een snelheid van ongeveer 60 characters per seconde. Als SAVE niet van een naam is voorzien wordt het programma zonder naam geschreven.
Aan de achterzijde van de computer zijn aansluitingen voor een kabel aanwezig, die verbonden moet worden met de microfooningang en de luidsprekeruitgang van de cassetterecorder.
Hier volgt een voorbeeld van het opbergen en weer teruglezen van een programma in de Teleac/DAI computer.

```
*LIST
10   REM TEST VAN HET WEGSCHRIJVEN NAAR DE CASSETTERECORDER
20   REM EN DAN WEER TERUGLEZEN.
30   REM TERWIJL INTUSSEN HET COMMANDO NEW WORDT GEGEVEN
40   REM OM AAN TE TONEN DAT ER ECHT WORDT GELEZEN.
*SAVE "VOORBEELD"
SET RECORD,START TAPE,TYPE SPACE
*NEW
*LIST
*LOAD "VOORBEELD"
*LIST
10   REM TEST VAN HET WEGSCHRIJVEN NAAR DE CASSETTERECORDER
20   REM EN DAN WEER TERUGLEZEN.
30   REM TERWIJL INTUSSEN HET COMMANDO NEW WORDT GEGEVEN
40   REM OM AAN TE TONEN DAT ER ECHT WORDT GELEZEN.
```

Eerst is het programma gelist, vervolgens geSAVED. De computer zegt na SAVE: SET RECORD, START TAPE, TYPE SPACE. De gebruiker moet dan de cassetterecorder op de gewenste plaats instellen, hem op record (ofwel opname) zetten en starten. Vervolgens moet een spatie worden ingetypt om de computer erop te wijzen dat nu het programma kan worden weggeschreven. Na het wegschrijven (onder de naam VOORBEELD in dit geval) geeft de computer weer de * om aan te geven dat we weer verder gaan en het opnemen moeten stoppen.
De gebruiker heeft de cassetterecorder daarna teruggespoeld naar het begin van de opname, op weergave gezet en vervolgens een LOAD-opdracht gegeven. Nota Bene: hierbij is een bandteller op de recorder van groot nut. Als men de plaats van een programma op de band opschrijft kan men dit programma snel weer terugvinden.
Intussen is nog even NEW en LIST ingetypt om het programma uit het geheugen te laten verwijderen en om aan te tonen dat dit ook is gebeurd.
Als de computer na LOAD weer met * terugkeert, wordt nogmaals LIST gegeven en zowaar: het programma is er weer. De cassetterecorder kan nu worden afgezet.
In een uitgebreidere versie van de Teleac/DAI personal computer is het mogelijk om het aan/uitzetten van de recorder door de computer te laten gebeuren. De afstandsbedieningsingang, die bedoeld is voor een schakelaartje op de microfoon van de recorder, wordt dan gebruikt. Het op opname of weergave zetten van de recorder moet nog steeds door de gebruiker zelf gebeuren.
De Exidy Sorcerer heeft standaard een aansluiting voor 2 cassetterecorders met start/stop mogelijkheden. Vrijwel iedere personal computer heeft hier iets andere mogelijkheden. Veelal is het niet mogelijk banden van het ene type computer op een ander type computer te verwerken.
Het gebruik van de cassetterecorder gaat dus met veel handelingen gepaard. De programma's worden in principe daar weggeschreven, waar de gebruiker de band heeft gezet. Als hij zich vergist, kunnen alle op de band aanwezige programma's worden overgeschreven. Zorgvuldigheid van handelen is dus wel vereist!
De snelheid van schrijven en lezen is betrekkelijk laag, omdat we in het beperkte frequentiegebied van de cassetterecorder moeten blijven. Een normale cassette kan 40 000 – 100 000 bytes bevatten. Schrijf/leessnelheid ligt tussen 110 en 1200 bits per seconde, afhankelijk van de gekozen personal computer.
Andere personal computers beschikken over vergelijkbare manieren om programma's en data op te bergen.

We hebben ons nu geconcentreerd op het opbergen en weer teruglezen van programma's. Maar programma's werken op gegevens (data). Daarom is het nodig deze data ook op te kunnen slaan op cassetteband. In plaats van SAVE moet dan het statement SAVEA worden gebruikt. In plaats van LOAD het statement LOADA. De toevoeging A slaat op Array. We kunnen op deze wijze een array-variabele of een gewone variabele opslaan en teruglezen. Wanneer we de array OUD naar cassette willen schrijven geven we in BASIC de opdracht:

```
10 SAVEA OUD "FILE"
```

Het saven gaat net als bij SAVE. De gegevens van de array OUD worden onder de naam FILE op cassette gezet.

20 LOADA NIEUW "FILE"

zal nu deze gegevens weer ophalen en toekennen aan de variabele NIEUW. De referentie loopt dus via de naam op cassette. Hier is dat dus "FILE". Het feit dat de gegevens oorspronkelijk vanuit de variabele OUD stammen wordt daarbij geheel vergeten.
Het aantal elementen van NIEUW moet daarbij kleiner of gelijk aan het aantal elementen van OUD zijn. Is dit aantal elementen kleiner dan worden de niet-gebruikte elementen vergeten (net als bij de DATA-READ combinaties). Zou het aantal elementen van NIEUW groter zijn dan volgt een foutmelding. Er mag ook een gewone variabele worden opgeborgen. String variabelen en string array's zijn eveneens toegestaan.



Met behulp van CHECK kunnen de namen van gegevens (programma's en data) van de cassette op het beeldscherm worden gezet. Dit geeft dus een soort inhoudsopgave weer. De Teleac/DAI personal computer zet bij een BASIC programma een 0 (nul) voor de naam, bij een array een 1 en bij een hexadecimaal gegeven en 2. We gaan in dit kader niet verder in op deze hexadecimale gegevens, die gewoonlijk een machinetaalprogramma vertegenwoordigen, zoals hoofdstuk 16 behandelt.

Een andere manier om programma's te laden is om (P)ROM's te gebruiken die het programma bevatten. Sommige personal computers beschikken daartoe over zogenaamde ROM-packs of Modules. Dit zijn ROM's die in plastic behuizing verpakt zijn. Deze behuizing kan dan in de computer worden gestoken. Zodra dit is gebeurd zal de computer het programma dat in de ROM's is opgeslagen uitvoeren. Het voordeel hiervan is, dat het programma niet eerst van cassette gelezen hoeft te worden. Enkele fabrikanten gaan daarbij zo

ver dat de gebruiker zelf zijn programmatuur in PROM's in de ROM-pack mag aanbrengen. Dan is het dus mogelijk volledig eigen programma's aan het systeem op te dragen. Men krijgt eigenlijk op deze wijze zijn eigen computers voor de eigen toepassing.
De meeste fabrikanten van personal computers bevelen een eenvoudige goedkope cassetterecorder aan. De aan zo'n recorder te stellen eisen zijn: een microfooningang met automatische volumecontrole: een koptelefoon- of extra luidsprekeruitgang met regelbaar volume en start/stopmogelijkheid met afstandsbediening. De uitgang van de personal computer wordt verbonden met de microfooningang van de cassetterecorder. De uitgang van de recorder gaat naar de ingang van de personal computer. Daarbij moet even worden uitgeprobeerd hoe het volume moet worden gezet. Gewoonlijk moet dit volume vrij luid staan. U moet meestal wel de luidspreker- of hoofdtelefoonuitgang gebruiken om voldoende volume te krijgen. Een gewone DIN-uitgang voor radioaansluiting geeft dan te weinig volume. De afstandsbediening van de computer wordt aangesloten op de start/stopmogelijkheid van de recorder. Deze start/stop is bij normaal gebruik meestal met een schakelaartje op de bijgeleverde microfoon aanwezig. Tenslotte is het erg prettig wanneer een bandteller aanwezig is. Dit vereenvoudigt het opzoeken van een programma op een band. In de praktijk blijkt het gebruik van één programma op één band veelal het handigst te werken, omdat de administratie daarvan eenvoudig is. Voor dit doel zijn er dan ook kortere bandjes in de handel. Sommige mensen gebruiken hun gewone spoelenbandrecorder voor bovengenoemd doel. Ook dit is een werkbare methode.

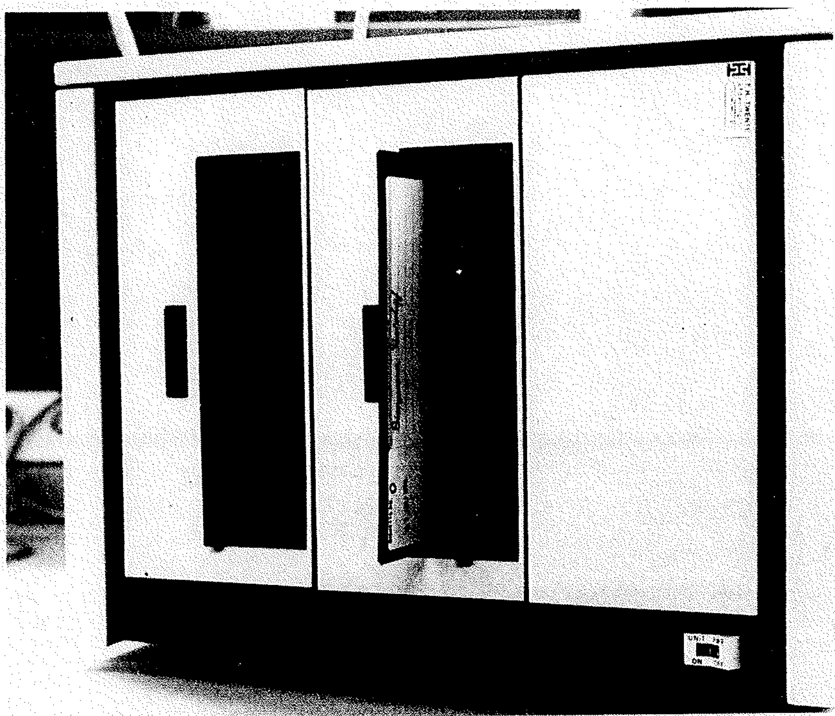
De recorders moeten wel een goede regelmatige bandsnelheid hebben. Mechanische betrouwbaarheid is uiteraard van belang. Minstens zo belangrijk als de recorder is de gebruikte cassetteband. Een betere band geeft hier de beste resultaten. Een heel goedkope band heeft vaak last van *dropouts*. Dit zijn onregelmatigheden in het magnetisch materiaal. Er zijn in de handel speciale (duurdere) cassettes die geen dropouts bevatten. Zij zijn bedoeld voor gebruik bij computers in digitale cassetterecorders. Ze lenen zich ook goed voor normaal gebruik.
Bij de cassettes moet men altijd oppassen bij het begin en het einde van de band. Daar is het magnetisch materiaal soms onregelmatig. Ook plaatsen waar veel gestopt wordt (begin, respectievelijk eind van een programma) zijn in dit opzicht verdacht. De band kan daar enigszins uitrekken. Let er vooral op dat u de band niet met uw vingers aanraakt, dit kan fouten veroorzaken. Spoel de bandjes dus steeds terug voordat u ze uit de recorder neemt.

Tenslotte: veel duurdere recorders beschikken over elektronische correctie van het geluid dat van de band komt. Daardoor wordt juist het signaal voor de computer verslechterd. Dus een eenvoudige recorder werkt beter. In de praktijk is gebleken dat een cassetterecorder met goede banden vrijwel nooit problemen oplevert.
Een verstandige methode is de programma's en data altijd twee keer op te slaan. Gaat er een keer iets mis dan blijft altijd nog de tweede versie over.
De genoemde automatische volumecontrole vereenvoudigt het ge-

bruik omdat dan alleen het volume voor de uitgang hoeft te worden ingesteld.
Er bestaan ook zogenaamde *digitale cassetterecorders*. Daarmee kan ook het heen- en terugspoelen vanuit de computer worden bestuurd. Dit bespaart veel handelingen. De snelheid van lezen en schrijven kan daarbij worden opgevoerd tot 5ØØ à 1ØØØ characters per seconde, omdat men niet per se meer in het hoorbare bereik hoeft te blijven. De prijs is veel hoger dan die van een gewone cassetterecorder.

19.6. Floppy disk

De cassetterecorder is in het dagelijks gebruik voor administratie en dergelijke nogal langzaam en vereist vele zorgvuldige handelingen. Een speciale recorder die vanuit de computer aan/uitgezet kan worden en op opname/weergave en dan ook nog met veel hogere snelheden werkt, zou ideaal zijn. Dit ideaal wordt in zekere zin bereikt met de floppy disk (zie afbeelding 19.6).



Afb. 19.6. Floppydisk eenheid

We hebben dan te maken met een ronddraaiende schijf (zoals een grammofoonplaatje) bedekt met magnetisch materiaal in plaats van met een band. Boven deze schijf zweeft een kop (weer zoals bij de grammofoonplaat). Via deze kop kunnen we onder besturing van de computer gegevens van de schijf lezen en ook er naar toe sturen. Tijdens lezen en schijven laten we de kop op de schijf rusten. We kunnen de kop ook naar binnen en naar buiten boven de schijf verplaatsen om gegevens te bereiken. Omdat floppy disks speciaal voor opslag van digitale gegevens geconstrueerd zijn, is de beper-

king, die in hoorbaar geluid lag weggevallen. De gegevens worden in cirkels op het oppervlak geschreven.
De hele besturing van zo'n floppy disk is in apparatuur en programmatuur een ingewikkelde zaak. De gemiddelde gebruiker zal dan ook van de leverancier een floppy disk-eenheid met de bijbehorende apparatuur en programmatuur voor aansluiting op zijn eigen systeem kopen. In Amerika is daarvoor het programma CP/M (Control Program/Monitor) het meest prominent. Veel verkrijgbare programmatuur is gebaseerd op CP/M. Wanneer u op dit moment dus CP/M als basis voor uw systeem kiest lijkt dit een verstandige keuze. U moet wel beseffen dat CP/M eenvoudiger is dan vergelijkbare programmatuur op grote computers. Dit geldt met name voor de mogelijke foutdetecties of foutmeldingen.
Hier is net zo min als voor de cassetterecorder een standaard gedefinieerd, niet in de apparatuur noch in de programmatuur.
Daarom geven we een summiere opsomming van de mogelijkheden, die ontstaan bij gebruik van een compleet systeem met floppy disk. Deze mogelijkheden vallen uiteen in 2 verschillende categorieën. De eerste omvat de hulp aan de bediener, die direct via het toetsenbord kan worden opgeroepen. De tweede is de hulp, die aan een gebruiksprogramma kan worden verleend.

19.7. Hulp aan de bediener

Vanuit het toetsenbord kunnen weer programma's onder naam op floppy disk worden opgeslagen en teruggelezen. Daarvoor bestaan weer opdrachten die we kunnen weergeven als SAVE en LOAD.
Het is mogelijk op te vragen welke namen met gegevens op de disk aanwezig zijn en hoeveel ruimte ze in beslag nemen. Het commando daarvoor heet vaak DIR (ofwel directory = catalogus).
Men spreekt bij een floppy disk over *files* die aanwezig zijn. Een file is een hoeveelheid gegevens die met behulp van een naam kan worden aangewezen. De gegevens kunnen betrekking hebben op een programma of op data.
Het is mogelijk een file van naam te doen veranderen met de opdracht REN (rename of verander van naam). Verder is het mogelijk files te verwijderen met de opdracht DEL (delete of verwijder).
Tenslotte is het mogelijk files naar een andere plaats op een floppy disk te kopiëren of naar een printer of iets dergelijks met het commando COP (COPY ofwel kopieer). Zo kan ook opdracht worden gegeven het programma te printen. Soms is het daarbij mogelijk meerdere files naar één resulterende file te kopiëren. Dan worden dus in feite files aan elkaar gekoppeld.
Vaak kunnen direct hulpprogramma's zoals *assemblers, editors* en *compilers* onder hun naam worden aangeroepen. Ook programma's die behulpzaam zijn bij het uittesten van programmatuur, kunnen op deze wijze worden gestart. Ook hier is er weer geen standaard aanwezig.

19.8. Hulp aan gebruikersprogramma's

De tweede categorie mogelijkheden is bedoeld om het de programmeur gemakkelijker te maken vanuit zijn programma met de randapparaten, waaronder floppy disk, te werken.

Daarbij horen de hulpprogramma's om data van en naar de verschillende randapparaten te halen en te zenden. De programmeur roept deze programma's vanuit zijn programma aan als subroutine en laat zo het systeem veel werk voor zich verrichten. Op die manier kunnen gemakkelijk gegevens naar printers, ponsbandponsers, enz. worden gestuurd. Gegevens van een toetsenbord, ponsbandlezer, enz. worden ook zo opgehaald.

We zullen nu vrij algemene voorbeelden kiezen, ontleent aan een floppydisk-systeem, zoals dat bij de TH Twente in de vakgroep Digitale Techniek ontwikkeld is.
Wanneer we te maken hebben met gegevens onder naam, krijgen we de mogelijkheid deze naam in ons programma te specificeren. Wanneer we een nieuwe file willen creëren roepen we een hulpprogramma CREATE (creëer) aan en geven de gewenste naam op. CREATE gaat na of de naam al op de floppy disk bestaat. Zo ja, dan meldt hij dit terug. Zo nee, dan schrijft hij de naam alvast weg naar de disk. Vervolgens kunnen we met een programma WRITE (schrijf) gegevens wegschrijven, die dan automatisch bij de zojuist opgegeven naam horen. Zijn we daarmee klaar, dan roepen we CLOSE (sluit) aan. Dan wordt de file afgesloten. Eventuele volgende gegevens gaan niet meer mee onder de hiervoor gebruikte naam.
Voor lezen hebben we de hulpprogramma's OPEN (ofwel zet klaar om te lezen) en READ (lees 1 character uit de geopende file). Daarbij wordt natuurlijk weer nagegaan of de file die we willen openen, aanwezig is. Zo niet, dan wordt een foutmelding gegeven.
De van het toetsenbord aanroepbare programma's zoals LOAD, SAVE, RENAME, enz. zijn meestal ook vanuit het gebruikersprogramma aan te roepen.
De hierboven gegeven beschrijving geeft een algemeen beeld van wat er zoal kan. Ieder specifiek systeem vertoont daarin zijn eigen specifieke eigenschappen en naamgeving.
De gemiddelde tijd om de kop op de juiste plaats te brengen, ligt in de orde van grootte van 1/4 seconde. De gebruiker hoeft bovendien niet meer telkens, als er gelezen of geschreven wordt, zelf iets te verzetten zoals bij de cassetterecorder het geval is.
De floppy disk is net als de cassette verwisselbaar. Er bestaan in de floppy disks vele uitvoeringen. De kleinste disks (vaak mini floppy's of mini diskettes genoemd) kunnen minimaal 89 ØØØ bytes (89 K) bevatten. Er bestaan ook grote floppy's met aan beide zijden een kop en speciale magnetische materialen, die ongeveer 1 2ØØ ØØØ bytes (1,2 M) kunnen bevatten.
De technologie en de prijs van floppy disk en de bijbehorende apparatuur is aan snelle verandering onderhevig. Er bestaan bijvoorbeeld al mini floppy's met 315 K opslagcapaciteit.
De lees/schrijfsnelheid van de floppy is veel groter dan die van een cassetterecorder. Een normale snelheid bedraagt als de kop reeds op de goede plaats is, 3Ø ØØØ tot 1ØØ ØØØ characters per seconde. Dit is dus veel sneller dan de gewone cassetterecorder, die globaal niet sneller dan ongeveer 120Ø characters per seconde is (12ØØ baud).
Bij de floppy disk moet goed op de juiste behandeling worden gelet. Vouwen, kreuken of stof kunnen lees- en schrijffouten veroorzaken. Vooral sigarettenrook en as zijn hierbij vaak de boosdoeners.
Tot nu toe is de floppy disk-eenheid ten opzichte van de computer

zelf erg duur, maar vaak een onmisbaar deel van het computersysteem.
Vooralsnog lijkt de floppy disk dus een ideaal medium voor gebruik in personal computers vanwege de snelle en grote opslagcapaciteit. De Teleac/DAI personal computer kan in principe via de 8255 die naar buiten gevoerd is, worden aangesloten op een floppy disk. Hier verdient het weer aanbeveling te wachten tot de fabrikant daardoor een compleet systeem levert.
Sommige andere personal computers zijn al voorzien van een of meerdere floppy disks of kunnen daarvan worden voorzien. Ook hier verdient het weer aanbeveling na te gaan wat er precies in zo'n systeem mogelijk is. Het software systeem CP/M kan daarbij als aanbeveling worden aangemerkt.

19.9. Overige I/O-uitbreidingen

Voor de meeste personal computers kunnen allerlei I/O-uitbreidingen worden gekocht. Daarbij valt te denken aan aanpassingen. Belangrijk hierin is dat ook analoog naar digitaal omzetters en digitaal naar analoog omzetters leverbaar zijn. Tenslotte kunnen veelal extra I/O-poorten worden aangesloten.
In de meeste gevallen moeten de personal computers dan uitgebreid worden met extra kaarten of zelfs met een extra kast. Slechts in enkele gevallen is uitbreiding mogelijk door rechtstreeks componenten in het al bestaande systeem bij te plaatsen.
Veel systemen leveren er ook zogenaamde *game paddles* bij. Dit zijn kastjes met een soort stuurknuppel. De stand van de stuurknuppel kan in de computer worden gelezen. Veel TV-spelletjes werken met deze game paddle, ook wel *joy-stick* genoemd. In de Teleac/DAI personal computer is er in BASIC zelfs het statement PDL (PaDdLe) aanwezig om de stand van een game paddle in te lezen. Het effect van 1Ø A = PDL (2) is dat A de positie van game paddle 2 krijgt. Deze positie kan van de waarde Ø tot 255 lopen. Dit correspondeert met de ruststand en de uiterste stand.
In de toekomst kunnen we ook bij personal computers vaste schijven verwachten met een opslagcapaciteit van 5 tot 8Ø miljoen characters. Voor vele administratieve toepassingen wordt dit ideaal. Waarschijnlijk zullen ook andere vormen van normaal geheugen worden gebruikt, zoals bubble memory (bellengeheugen). Daarbij kunnen we draagbare apparaten verwachten met een grote opslagcapaciteit.
Voor juiste details raadplege men de bij een systeem behorende handleiding.
Opmerking: Qua prijs zijn de randapparaten veel duurder dan de computer zelf. Verstandig kiezen is hier absolute noodzaak.

Samenvatting hoofdstuk 19

In dit hoofdstuk hebben we een beeld kunnen krijgen van wat er in machinetaal nodig is om een eenvoudige printer aan te sluiten. Daarna is in het algemeen ingegaan op het aansluiten van een printer, een toetsenbord, een modem, een gewone cassetterecorder en een floppy disk.

Hoofdstuk 20

Slot

In dit laatste hoofdstuk gaan we in op vragen als: wat hebben we nu eigenlijk geleerd; wat kunnen we echt met de Teleac/DAI-computer; welke andere microcomputers zijn er en ook hoe de toekomst er op dit gebied uitziet.
In Microprocessors 1 *en* 2 *hebben we geleerd hoe een microcomputer wordt opgebouwd. Als voorbeeld is de 8080 microprocessor gebruikt. De 8080 chip is van groot belang omdat er zeer veel toepassingen zijn, waarin de 8080 is ingebouwd. Er is bovendien nog een grote categorie microcomputers, die sterk op de 8080 lijken.*

20.1. Computeropbouw

We zijn ingegaan op de basisopbouw van de microcomputer zonder dat we nu in staat zijn om er direct zelf een te bouwen. De details van de daartoe benodigde componenten zijn niet behandeld. Evenmin is ingegaan op de aspecten, die te maken hebben met voeding, ontstoring en behuizing. Dit zou ons ver in de techniek hebben gevoerd. Het doel van dit boek was immers een inzicht en een overzicht te geven aan een zo breed mogelijk publiek.
De tendens in microcomputers is er sterk op gericht om kant-en-klare systemen te kopen en die dan, indien nodig, voor een toepassing te programmeren. Daarbij speelt dan nog wel het aspect van de aanpassing een rol. Meetopnemers voor spanning, temperatuur of druk kunnen benodigd zijn. De programmering heeft zoveel aandacht gekregen, dat men daarmede zelfstandig vooruit kan. Dat wil zeggen, men kan zelf verder oefenen.
De aanpassing is nauwelijks behandeld. Daar zal dit boek voor de specifieke toepassing geen houvast geven. Het zou ook te gemakkelijk gesteld zijn om aan te nemen dat men nu grote softwarepakketten kan ontwikkelen. Daarvoor is een veel uitgebreidere vaardigheid met het ontwikkelen van programma's vereist. Wel heeft men elementair begrip gekregen van wat mogelijk en onmogelijk is. Men moet nu in staat zijn eenvoudige programma's zelf op te zetten, te programmeren en te documenteren. Bij veel toepassingen lossen kleine programma's het probleem gelukkig op. Het is uiteraard niet zo, dat degenen die *Microprocessors 1 en 2* bestudeerd hebben nu computer-specialisten zijn geworden. Zij hebben wel enig inzicht gekregen. Ervaring ontbreekt evenwel nog. Deze ervaring kan worden opgebouwd door zelfstandig oefenen. Nog beter kan dit zelfstandig oefenen worden gedaan aan de hand van ervaren deskundigen of goede cursussen. Een toepasselijke uitspraak van Joseph Weisen-

baum hier is: 'Voor wie alleen een hamer heeft lijkt de hele wereld op een spijker. Voor wie alleen een computer heeft lijkt de hele wereld op een computersysteem'.

20.2. Instructieset

Over het algemeen zijn de instructiesets van andere microprocessors niet zo erg afwijkend van de 8080 instructieset. Met de kennis van de 8080 chip moet men redelijk snel in staat zijn ook de andere instructiesets te begrijpen en te programmeren. Het enige probleem, dat daar zou kunnen optreden is, dat de 8080 een nogal beperkt aantal adresseringsformaten kent zoals directe adressering, indirecte adressering en immediate adressering.
Sommige andere microcomputers zijn daarin ruimer voorzien.
Wat het aantal bruikbare registers voor de programmering betreft, bestaan er zowel microcomputers met meer registers als met minder. De 8080 neemt daar een soort middenpositie in.
Voor de I/O kent de 8080 afzonderlijke instructies. Verschillende andere processors werken alleen met memory-mapped mode, waarbij met gewone verplaatsinginstructies naar de buitenwereld wordt gegaan. Dan moet wel zorgvuldig worden nagegaan of ook de andere voor het geheugen gedefinieerde instructie op de I/O-poorten goed werken.
De exacte benamingen van instructiecodes zijn per microprocessor verschillend. Voor de meest gangbare 8 bits microcomputers kan met ongeveer 7Ø verschillende instructies de hele instructieset worden beschreven. De onderlinge namen kunnen verschillen, de onderlinge mogelijkheden zijn niet sterk afwijkend. Globaal is het zo dat de uitspraak: 'Een doorsnee toepassing kan met iedere microprocessor gerealiseerd worden' de waarheid dicht benadert. Alleen in speciale toepassingen is de ene microprocessor soms duidelijk beter dan de andere, bijvoorbeeld om snelheidsredenen.

20.3. Andere microcomputers

Als belangrijkste concurrenten van de 8080 microcomputers kunnen we noemen:
- MOTOROLA 68ØØ, die vergelijkbaar is, maar iets meer adressering modes kent.
- ZILOG Z8Ø, een duidelijk uitgebreidere 8080.
- MOS TECHNOLOGY 6502, sterk op de 68ØØ lijkend.
- INTEL 8085, een 8080 die minder componenten vergt.
- FAIRCHILD F8 met een eenvoudiger instructieset.

We gaan hier niet alle microcomputers opsommen. Er bestaan naar schatting nu al meer dan 2ØØ verschillende typen. We willen wel wat lijnen van microcomputers uitzetten.
Veel eenvoudige regeltoepassingen vereisen relatief weinig rekenwerk, weinig geheugen en weinig I/O. Dit gaat met name op voor toepassingen in de huishoudelijke sfeer. Hier zijn vaak 4 bits microcomputers in gebruik, die met 1 of 2 chips een heel systeem kunnen bouwen. De meest prominente is de TEXAS INSTRUMENTS TMS1ØØØ.

Tegenwoordig zijn er ook veel 8 bits microcomputers op 1 chip te koop voor eenvoudige toepassingen. Enkele voorbeelden daarvan zijn: de INTEL 8048, MOTOROLA 68Ø1 en de MOSTEK F8. Deze treffen we dan ook in eenvoudiger toepassingen aan. Uitbreiding voor grotere toepassingen is moeilijk of onmogelijk. Zeer prominent is een zodanige keuze van chips, dat met 2 chips (bijvoorbeeld CPU en ROM + I/O) een microcomputer kan worden gebouwd. Het is dan mogelijk voor uitgebreidere toepassingen gewoon extra chips toe te voegen. Deze klasse microcomputers lijkt zich duidelijk in een toenemende belangstelling te kunnen verheugen. Een voorbeeld hiervan is de INTEL 8055 en de F8.

Een belangrijke tendens is die naar 16 bits microcomputers. De rekencapaciteit daarvan is in principe beter dan bij de 8 bits microcomputer. Sommige van deze 16 bits microcomputers richten zich duidelijk op het vervangen van vergelijkbare 16 bits minicomputers. Zulke minicomputers worden al jaren toegepast in vele technische en administratieve systemen. Voorbeelden van deze microcomputers zijn: de LSI 11 van DIGITAL EQUIPMENT en de MICRO NOVA van DATA GENERAL.
Daarnaast verschijnen er 16 bits microcomputers, die geen minicomputer-equivalent kennen. Hier zijn vooral de INTEL 8086 en de MOTOROLA MC68000, de ZILOG Z8000 en de TEXAS INSTRUMENTS TI9900 van belang. Het lijkt er op dat deze microcomputers in de toekomst een groot marktaandeel in die toepassingen zullen verwerven, waar veel moet worden gerekend.
De snelheid, waarmee de technologie voortschrijdt, doet de vermoedens rechtvaardigen dat in de nabije toekomst ook de heel grote computers als chips met grote geheugencapaciteit en groot rekencapaciteit ter beschikking komen. Dit opent de weg naar nog veel meer toepassingsmogelijkheden. Voor de industriële gebruikers is het een belangrijke zaak die toepassingsmogelijkheden te benutten. Slechts aan een zeer klein deel van de mogelijke toepassingen wordt momenteel serieus gewerkt, zodat hier nog vele mogelijkheden liggen.

20.4. Maatschappij

Bij dit geheel treden onvermijdelijk vragen op omtrent de wenselijkheid van deze ontwikkeling en de maatschappelijke gevolgen. Deze vragen kunnen we hier niet beantwoorden. We hebben wel het gevoel dat boeken als dit de discussie over de maatschappelijke aspecten kunnen verbeteren. Hoe meer de kennis over een onderwerp verspreid is, des te zinniger kan men over dat onderwerp praten.
De microcomputer zal een uiterst belangrijke rol spelen in onze toekomstige samenleving. Aan de ene zijde gaat er een grote dreiging vanuit vanwege de grote bedreiging voor de werkgelegenheid en de privacy van individuele burgers. Anderzijds bestaat er de mogelijkheid dat werk gecreëerd wordt en dat een bijdrage aan een meer leefbare wereld ontstaat. Zekerheid over de richting, waarin zich dit ontwikkelt, is er bepaald niet. Uiteraard geeft ook dit boek geen oplossing daarvoor. Wel wordt er een bijdrage geleverd in de vorm van enig begrip voor de technische mogelijkheden van microcom-

puters. Dit is op zich reeds belangrijk als men hoort en leest welke nonsens vaak in de discussies op dit gebied naar voren wordt gebracht.

20.5. BASIC

In dit boek hebben we ook BASIC leren programmeren. Dan is het in feite onbelangrijk welke microcomputer in het systeem verborgen zit. Het enige verschil dat de gebruiker merkt is dat zijn programma sneller kan worden uitgevoerd in een systeem met een krachtigere onderliggende microcomputer. De huidige ontwikkelingen wijzen erop dat BASIC de standaard voor hogere orde talen voor personal computers is geworden. De oorzaak daarvan is gelegen in het feit dat het betrekkelijk eenvoudig is een BASIC interpreter te maken. Veel software in BASIC is reeds aanwezig op grotere computers. Ook de eenvoud van BASIC heeft hier stimulerend gewerkt. We moeten wel bedenken dat sommige andere talen meer structuur aan programma's opleggen. Er zijn veel dialecten van BASIC in de handel, met onderlinge kleine verschillen. Het omzetten van een BASIC programma van het ene systeem naar het andere levert bijna nooit grote problemen. In dit boek zijn van verschillende personal computers bijlagen toegevoegd om de verschillen tussen deze computers en de Teleac/DAI personal computer aan te geven. Wanneer men over zo'n systeem beschikt zal men dan ook dit boek goed moeten kunnen volgen. Belangrijke verschillen in BASIC tussen de verschillende computers zijn onder andere op de volgende punten te vinden.

- Het feit of strings toegelaten zijn.
- Het feit of strings met meerdere dimensies zijn toegestaan.
- De toegestane lengte van namen van variabelen, zodat deze namen als verschillend herkend worden.
- De rekennauwkeurigheid.
- De naamgeving van de functies en de exacte definitie.
- De mogelijkheid voor definitie van eigen functienamen.
- De mogelijkheid voor aanroep van machinetaalprogramma's.
- De mogelijkheid voor printen volgens voorgeschreven formaat.
- De mogelijkheid voor geheugen en I/O-benadering.
- De grafische mogelijkheden.
- De mogelijkheid voor geluid.
- De mogelijkheid voor direct gebruik van spraak.
- De uitbreidingsmogelijkheden voor onder andere printers, floppy disks, I/O, geheugen, enz.
- De mogelijkheden voor file I/O op cassette en floppy disk.
- De verkrijgbare programmatuur!!!!
- De editing-faciliteiten voor intypen en verbeteren van BASIC programma's, bij voorbeeld met CURSOR besturing.
- De mogelijkheid om commando's en functies met verkorte naamgeving in te typen en op te slaan (bijvoorbeeld 1 of 2 letters), terwijl met volle naam gelist wordt.
- Automatische regelnummering en/of automatische hernummering.

20.6. Andere talen

De programmeertaal PASCAL gaat in de toekomst waarschijnlijk eveneens een belangrijke rol spelen. Een groot nadeel van PASCAL is dat deze op 8 bits microcomputers meestal niet erg efficiënt is en derhalve traag. Voor veel toepassingen is dat echter niet bezwaarlijk. PASCAL heeft een goede kans een belangrijke standaardtaal te worden.
Voor 16 bits microcomputers lijkt PASCAL de juiste keuze. Het grote voordeel van PASCAL is dat de taal tot het schrijven van beter gestructureerde programma's leidt. De overdracht en de geldigheid van parameters over programmadelen is goed gedefinieerd. Labels zijn toegestaan en de taal heeft een duidelijk gedefinieerde structuur. Er staan in PASCAL verschillende zeer krachtige manieren ter beschikking voor de behandeling van data. Er zijn op dit moment echter nog maar weinig microcomputers, die de taal PASCAL via een compiler of interpreter ondersteunen. De compiler vereist namelijk veel geheugen. Met name in de ontwikkeling van de 16 bits microcomputer zal PASCAL echter aan belangrijkheid winnen. Het op de universiteit van San Diego ontwikkelde UCSD PASCAL vindt steeds meer ingang. Dit systeem combineert PASCAL en het operating systeem in één.

Vanuit de grote computers en de minicomputers is veel programmatuur ter beschikking in talen als FORTRAN (rekenwerk gericht) en COBOL (administratief gericht). Ook deze talen manifesteren zich in de microcomputersfeer. In feite kunnen we alle belangrijke talen van grote computers terug verwachten voor microcomputers. Daarbij zullen zij dan ondersteund moeten worden door voldoende krachtige systeemsoftware.
Men verwacht ook een belangrijke invloed van de taal ADA die door het Amerikaanse Ministerie van Defensie tot standaard is gekozen.

20.7. Teleac/DAI personal computer

Ten aanzien van de Teleac/DAI personal computer kunnen we opmerken dat deze qua prijs en mogelijkheden een uniek produkt is. De computer bevat alle noodzakelijke mogelijkheden voor een breed vlak van toepassingen. Voor studiedoeleinden, huishoudberekeningen en huishoudelijke besturing is de Teleac/DAI personal computer zonder meer uitstekend bruikbaar. Voor administratief werk is al snel de uitbreiding met floppy disk gewenst, zodat sneller toegang tot gegevens kan worden verkregen. Niet alleen de snelheid speelt een rol, maar ook de grotere hoeveelheid van gegevens, die op de floppy kunnen worden opgeslagen.
Een tweede voor de hand liggende uitbreiding is een printer om de resultaten op papier vast te leggen.
Helaas is de prijs van floppy disk en printer veel hoger dan de prijs van de personal computer. Toch zal een systeem met deze 3 componenten voor veel administraties al binnen de mogelijkheden liggen. Het enige dat dan nog ontbreekt is standaard toepassingsprogrammatuur. Dit zal door de leverancier moeten worden bijgeleverd en wel voor de Nederlandse situatie.
Voor groter administratief werk is op dit moment de verstandigste

keuze nog steeds een speciaal doelgericht systeem met een harde disk als opslagmedium. De ontwikkelingen in de computers gaan zo snel dat we nog zeer veel toepassingen zullen ontmoeten. Een nadeel voor eenvoudiger systemen is gelegen in het gebrek aan betrouwbaarheid van programmatuur en apparatuur. Een andere tekortkoming is het gebrek aan voldoende achtergrondgeheugen.

In dit boek is de Teleac/DAI personal computer steeds uitdrukkelijk als voorbeeld genomen. Naar inzicht van de auteurs is het beter één voorbeeldmachine goed uit te diepen dan van vele voorbeelden steeds weinig te laten zien.
Soms is daarbij nogal diep op de specifieke details van de Teleac/DAI personal computer ingegaan. Deze details dragen dan bij tot een beter begrijpen van andere systemen. Bij deze andere systemen zullen die details vaak iets anders liggen. Wanneer men echter niet ook eens in de details heeft gekeken kan men dikwijls niet precies begrijpen wat er aan de hand is. Met de bijlagen over andere computers is dit probleem overigens ten dele opgelost. Deze bijlagen vallen geheel onder verantwoordelijkheid van de leveranciers. De redactie heeft hierin niet bijgedragen.

De mogelijkheden van de Teleac/DAI personal computer zijn zo groot dat de meeste andere personal computers minder faciliteiten bezitten. Op sommige plaatsen in de tekst zijn mogelijkheden van andere computers aangestipt, daar waar de Teleac/DAI personal computer die niet kende; een slechts weinig voorkomende situatie. Gewoonlijk zal het zo zijn dat een personal computer eenvoudiger is dan de Teleac/DAI personal computer. Als redactie hebben we gemeend dat daarom deze computer een ideaal voorbeeld is. Eventuele andere computers zijn dan gemakkelijk te begrijpen. Men kan de hier behandelde, maar in zijn computer niet aanwezige mogelijkheden gewoon vergeten. Deze situatie leek ons wenselijker dan een voorbeeldmachine, die zo eenvoudig is dat vele zaken niet aan de orde komen. De voorbeelden zijn alle afgedrukt zoals ze op de Teleac/DAI personal computer verschijnen. Er is daarbij naar een maximale herkenbaarheid gestreefd. Veel hoofdstukken zijn nog met een prototype computer geproduceerd. Het kan dus voorkomen dat er kleine afwijkingen bij de uiteindelijke computers optreden. We stellen het op prijs wanneer u die aan Teleac wilt melden.
BASIC is in de Exidy Sorcerer eenvoudiger dan in de Teleac/DAI computer. De Sorcerer als algemene personal computer is een goede keuze omdat de computer robuust is en reeds veel randapparatuur en programmatuur als ondersteuning heeft. De Sorcerer is als zodanig een waardig vervanger voor de Teleac/DAI computer.

Wanneer de lezer dit boek geheel heeft doorgenomen, zal hij/zij in staat moeten zijn zelfstandig de handleidingen van andere personal computers te begrijpen en te beoordelen. Ten dele ziet dit boek er dientengevolge ook uit zoals een handleiding van een computer er uit zou moeten zien.

Verklarende Woordenlijst

Access
toegang tot geheugen of I/O

Accumulator
een register in het rekenorgaan (ALU) in een computer waarop rekenkundige en logische bewerkingen worden uitgevoerd.

Adresbus
een verzameling draden waarover de processor adressen kan overdragen naar geheugen en/of I/O.

Adresregister
een register waarin het adres van de volgende memory- of I/O-access staat.

Algorithme
een beschrijving van een gedragspatroon in ondubbelzinnig benoemde acties.

ALU
Arithmetic and Logic Unit: een subsysteem van een computer dat rekenkundige en logische bewerkingen uitvoert.

Analoog
continu verlopend.

Applicatieprogramma
een programma geschreven voor een bepaalde specifieke toepassing.

Architectuur
de architectuur van een computer geeft het uitwendige gedrag van die computer weer.

Array
een reeks bij elkaar behorende en in een logische volgorde geplaatste gegevens (meestal getallen).

ASCII
American Standard Code for Information Interchange: een standaard code voor informatie-uitwisseling.

Assembleertaal
een machinegerichte computertaal.

Assembler
een vertaalprogramma dat een in een symbolische taal geschreven programma omvormt tot een programma in machinetaal. Dit gebeurt door symbolische operatiecodes en adressen om te zetten in binaire getallen.

Assembleren
het omzetten van symbolische instructies in binaire getallen.

Auxiliary carry
zie intermediate carry.

BASIC
Beginners All purpose Symbolic Instruction Code; een eenvoudige programmeertaal die veel overeenkomst vertoont met de algebra.

Baud
aantal bits per seconde. Een maat voor de transmissiesnelheid bij seriele overdracht.

BCD
Binary Coded Decimal: binaire codering van de decimale cijfers Ø t/m 9 d.m.v. 4 bits.

Bi-Directioneel
werkend in 2 richtingen.

Binair getal
een getal dat is opgebouwd uit nullen (Ø) en enen (1).

Bit
BInary digiT: een eenheid van informatie, een Ø of een 1.

Bit sliced processor
een uit moten opgebouwde processor waarbij elke moot een woordlengte van verscheidene bits bezit.

Blokschema
zie flowchart.

Borrow
het bij aftrekken lenen van een cijfer uit een hogere macht.

Bubble sort
een sorteermethode waarbij de te sorteren elementen steeds van plaats verwisselen met elementen van een lagere rangorde.

Byte
een groep van 8 aaneengesloten bits.

Carry
een overdracht die kan ontstaan bij rekenkundige bewerkingen (1 onthouden).

Character generator
een circuit waarin de vorm van elk teken is opgeslagen.

Chip
het materiaal waarop de elektronica van een schakeling gebouwd is.

Concatenation
aaneenschakeling; het samenknopen van strings.

Conditiecoderegister
zie statusregister

Console
het bedieningspaneel van een computer waarmee de CPU te besturen en te controleren is.

Controlbus
een groep draden waarover besturingssignalen verzonden worden.

CPU
Central Processing Unit: ofwel Centrale Verwerkings Eenheid (CVE) van de computer.

Current loop
een communicatieverbinding waarbij een elektrische stroom voor het datatransport zorgt.

Cursor
de 'aanwijsstok' die op het T.V.-scherm aangeeft waar het volgende teken geschreven kan worden.

Data
gegevens, informatie.

Data-acquisition
het binnenhalen, converteren naar digitale vorm en verwerken van analoge gegevens.

Databus
een groep draden waarover de gegevens (data) in een computersysteem getransporteerd worden.

Dataregister
een register waarin data wordt opgeslagen.

Debuggen
het opsporen en veranderen van fouten in software of hardware van een computer. Ook wel genoemd 'ontluizen'.

Digitaal
werkend met discrete waarden.

Directieven
speciale assembler commando's bedoeld voor de assembler zelf.

DMA
Direct Memory Acces: de randapparaten hebben bij DMA direct toegang tot het geheugen buiten de processor om.

Editor
een hulpprogramma voor het intypen en corrigeren van tekst en sourceprogramma's.

EPROM
Erasable Programmable Read Only Memory: een programmeerbaar leesgeheugen dat d.m.v. ultraviolet licht te wissen is, en daarna opnieuw geprogrammeerd kan worden.

Executie
het in werking zijn van een programma of een deel daarvan.

Executie cycle
dat deel van de instructie waarin de gespecificeerde operatie wordt uitgevoerd.

False
onwaar, nee.

Fetch cycle
dat deel van de instructie waarin de gespecificeerde operatie wordt ingelezen en gedecodeerd.

Fixed point
vaste komma; methode van vastlegging en verwerking van getallen, waarbij de cijfers voor en na de komma hetzij afzonderlijk worden aangeduid, hetzij impliciet op een vaste plaats van elkaar zijn gescheiden.

Flag
een indicatie in een computer, meestal een bit in het status register, dat aangeeft hoe een bepaalde operatie is uitgevoerd.

Floating point
zwevende komma; methode van vastlegging en verwerking van getallen waarbij een getal wordt gesplitst in een fractie en een radix vermenigvuldiger. Voorbeeld: 2530 = 0,253 * 10 ↑ 4
Deze beide elementen (0,253 en 10 ↑ 4) ondergaan in de berekening afzonderlijke bewerkingen.

Floppy disk
een flexibele, kleine schijf die gebruikt kan worden als opslagmedium voor gegevens.

Flow-chart
stroomschema: een symbolische voorstelling die alle logische stappen van een computerprogramma toont.

Fractie
dat deel van een getal dat zich na de komma bevindt. Voorbeeld: getal = 25,36 fractie = 0,36.

Full duplex
een communicatiemethode waarbij gelijktijdig data verzonden en ontvangen kan worden.

General purpose register
een register voor algemeen gebruik.

Half duplex
een communicatiemethode waarbij slechts in één richting tegelijk data verzonden of ontvangen kan worden.

Handshaking
een methode waarmee gegevens op een betrouwbare wijze kunnen worden overgedragen. De gegevens worden begeleid door ten minste twee besturingssignalen, de zgn. handshake signalen.

Hardcopy
afdruk in de vorm van een printout of fotocopie.

Hardware
de mechanische, magnetische of elektrische delen waaruit een computersysteem is opgebouwd.

Hexadecimaal
talstelsel met een radix (grondtal) van 16. Om de 16 mogelijke codes weer te geven worden de cijfers Ø t/m 9 en de letters A t/m F gebruikt.

Identifier
identificatiesymbool; een teken dat wordt gebruikt om een aantal bij elkaar behorende gegevens te identificeren, aan te duiden of van een naam te voorzien.

Immediate adressering
de operand van een instructie is in de instructie zelf aangegeven.

Impliciete adressering
het adres van een operand is in de instructie bekend verondersteld.

Index
een aanwijzer; een teken of een getal dat bij een symbool wordt gevoegd om het van andere gelijke symbolen te onderscheiden. (bijv. A1, A2, A3, An)

Indirecte adressering
het adres van de operand in een instructie wordt via de inhoud van een aangegeven adres gevonden.

Input
invoer die vanuit de buitenwereld wordt overgebracht naar het interne geheugen van de computer, ter verwerking door de CPU.

Instructie register
het register in de CPU dat de instructiecode bevat.

Instructie set
het totaal aan instructies die, in machinetaal, door een bepaalde computer (soms een bepaalde versie daarvan) direct kunnen worden uitgevoerd.

Integer
een geheel getal bijv. 25.

Interactief
een computer systeem waarbij commando's of programma's direct uitgevoerd worden, dit in tegenstelling tot een batch systeem waarbij de te verwerken gegevens opgespaard worden en achter elkaar gedurende een enkele machine run verwerkt worden.

Interface
de koppeling tussen systeemdelen van een computer. Meestal tussen randapparaten en computer.

Intermediate carry
hulpcarry, een carry in het midden van een byte. Wordt gebruikt voor BCD berekeningen.

Interpreter
een programma dat statement voor statement een hogere programmeertaal executeert.

I/O
Input/Output ofwel invoer/uitvoer.

I/O-mapped I/O
de I/O poorten zijn zodanig geschakeld dat ze gelezen of beschreven kunnen worden via speciale I/O instructies.

Keyboard
toetsenbord waarmee een randapparaat kan worden bediend of waarmee gegevens kunnen worden ingevoerd.

Label
een symbolische adresaanduiding in een programma.

Literal
zie immediate.

LSI
Large Scale Integration: geïntegreerde schakelingen met hoge pakkingsdichtheden, of wel zeer veel componenten op een klein oppervlak.

Magnitude
grootte; gewoonlijk in de term 'sign and magnitude', tekst en grootte.

Memory map
een lijst waarin vermeld staat welke geheugen gebieden gebruikt worden door bepaalde programma's en welke nog ongebruikt zijn.

Memory-mapped I/O
de I/O poorten zijn zodanig geschakeld dat ze gelezen of beschreven worden via memory access instructies.

Microprocessor
een enkel geïntegreerd circuit dat een groot deel van de kracht van een kleine computer bezit.

Microprogramma
in de CPU opgeslagen programma dat de besturing van de instructies regelt.

Minicomputer
een computer waarvan in de meeste gevallen de woordlengte 16 bits (of meer) bedraagt.

Mnemonic
instructie voor een computer, geschreven op een wijze die door de programmeur gemakkelijk kan worden onthouden. SUB voor SUBTRACT. Deze mnemonic moet later wel in machinecode worden omgezet.

Modem
MOdulator-DEModulator; een apparaat dat signalen van en naar de computer (bijv. over een telefoonlijn) converteert om als koppeling te fungeren tussen de verschillende delen van een terminalsysteem.

Monitor
een programma, meestal een deel van het operating systeem dat de timing, de in- en uitvoer en communicatie met de gebruiker regelt.

MSI
Medium Scale Integration; geïntegreerde schakelingen met een lagere pakkingsdichtheid dan bij LSI.

Multiple branch
meervoudige sprong; een instructie waarbij het volgende adres waar de computer verder gaat afhankelijk is van de waarde van een expressie.

Nibble
een groep van 4 aaneengesloten bits.

Numeriek
een term die wordt gebruikt om aan te geven dat een gegeven geheel uit cijfers bestaat.

Object code
een programma dat direct door de computer kan worden uitgevoerd.

Octaal
talstelsel met een radix (grondtal) van 8. De gebruikte symbolen zijn de cijfers Ø t/m 7.

One chip computer
een complete computer op 1 circuit (chip).

Operand
de grootheid die wordt bewerkt door de computer.

Operatiecode
deel van een instructie code die aangeeft welke operatie moet worden uitgevoerd.

Operatoren
een operator voert op één of meer operanden een bewerking uit. Voorbeeld van een operator is +, −, *, /

Output
informatie, verwerkt door de CPU, die wordt overgebracht van het interne geheugen van de computer naar de buitenwereld.

Overflow
de toestand die optreedt als het resultaat van een rekenkundige bewerking buiten de capaciteit van de computer valt.

Pariteitsbit
een extra bit dat toegevoegd wordt aan een computerwoord om een foutcontrole mogelijk te maken. Bij een even pariteit is de som van alle bits die hoog zijn en het pariteitsbit een even getal.

Periferie
randapparatuur of I/O apparatuur die onder controle van de CPU bestuurd kunnen worden. Enkele voorbeelden: kaartlezer, kaartponser, floppy disk drive, printers.

Programmateller
register dat het geheugenadres van de volgende instructie, die in het computerprogramma moet worden uitgevoerd bevat.

Protocollen
afspraken over de vorm en inhoud van gegevensoverdracht.

RAM
Random Access Memory: een lees- en schrijfgeheugen.

Random
willekeurig, toevallig.

Real
een gebroken getal bijv. 12,36.

Register
een hulpgeheugen dat een speciale functie heeft in de computer, en gewoonlijk verkort geadresseerd wordt.

Reset
herstellen; op nul zetten.

ROM
Read Only Memory; leesgeheugen met een vaste inhoud.

RS 232
een norm voor serieel datatransport.

Scalair
een enkelvoudige variabele zonder kental.

Sequentieel
achtereenvolgend.

Sign
teken; hiermee worden negatieve getallen onderscheiden van positieve getallen.

Software
de programma's voor een computersysteem.

Source code
het programma dat als invoer voor de assembler dient.

Stack
een gebied in het geheugen waarin tijdelijk o.a. registerinformaties en terugkeeradressen van subroutines worden opgeslagen. Werkend volgens het mechanisme van laatst in, eerst eruit.

Stackpointer
een register dat de momentane plaats van de stack in het geheugen adresseert.

Statement
opdracht; een zinvolle combinatie van termen of een instructie in een programmeertaal.

Statusregister
de inhoud van de interne registers samen met de conditieflags. Deze geven tijdens de

programma-executie de toestand aan, waarin een computer zich bevindt.

Stored program techniek
het programma in het computergeheugen opgeslagen.

String
een rij characters.

Stroomschema
zie flow-chart.

Subroutine
een subprogramma dat niet in het hoofdprogramma is opgenomen. Een subroutine wordt via een speciale aanroep uit het hoofdprogramma aangeroepen.

Subscript
zie index.

Support software
hulpprogramma's die bij een computersysteem geleverd worden.

Synchronisatie
het proces van het op elkaar afstemmen van de frequentie en de fase van twee of meer signalen.

Syntax
taalkundige definitie van commando's of statements.

Systeemprogramma's
de programma's die standaard bij een computersysteem geleverd worden.

Systeemterminal
een terminal die rechtstreeks met de computer verbonden is en waarmee het computersysteem opgestart kan worden.

Terminal
een input of output apparaat waar gegevens een computersysteem verlaten of binnenkomen. (bijv. teletype of een video terminal).

Terminator
scheidingsteken.

Tetrade
de helft van een octade (= byte).

Time sharing
tijd verdelen; het verdelen van de CPU tijd in kleine stukjes over de verschillende programma's die tegelijk aangeboden worden aan de computer. Hierdoor lijkt het dat de verschillende programma's simultaan verwerkt worden door de computer.

Tracer
een hulpprogramma dat een gebruikersprogramma stap voor stap laat executeren en daarbij een overzicht van de registerinhouden en de programcounter laat zien. Hierdoor is het eenvoudig een programma te debuggen.

True
waar, ja.

Turn-key systeem
een computer dat een probleem geheel oplost. De gebruiker hoeft het systeem slechts aan te zetten.

Two complement
een afspraak over de manier waarop berekeningen in een computer worden geïnterpreteerd.

UART
Universal Asynchronous Receiver Transmitter; een I/O bouwsteen die de asynchrone seriële datatransmissie verzorgt.

USART
Universal Synchronous/Asynchronous Receiver Transmitter; een I/O bouwsteen voor synchrone en asychrone seriële datatransmissie.

Utility programma
een standaardprogramma of routine dat de werking van een computer ondersteunt. Bijvoorbeeld een conversie- of sorteerprogramma.

Video display
een apparaat waarbij de informatie getoond wordt op een beeldscherm.

VLSI
Very Large Scale Integration: integratie op zeer grote schaal, bijv. een microcomputer op 1 chip.

Zero flag
een schakeling, die de logische waarde 1 krijgt indien het resultaat van een instructie de waarde nul heeft.

Uitwerkingen antwoorden multiple choice-vragen

Hoofdstuk 3

A.3 **a** BASIC constanten mogen geen komma bevatten
b Gebroken getallen noteren als: gehele deel — punt — fractie; dus 0 . 5
c De macht van 10 is in BASIC altijd een integer (geheel getal)
d Het moet zijn: — 0.123 E — 11

B.1 Een identifier kan alleen opgebouwd zijn uit:
— letter(s)
— letter(s) cijfer
— letter(s) $

C.4 Uitspraak a: zie antwoord **A. a.**
Uitspraak b: zie § 2.3

D.3 **b** Moet zijn: C = (A ↑ 2 + B ↑ 2) ↑ 0.5
c Moet zijn: R = (P — Q) ↑ (S — 3)
d Moet zijn: 7.0 * X ↑ (1.0/6.0) want er staat 7 * x⅙ en niet $7^{1/6}$ * $x^{1/6}$

E.2 T_1 = A * B/2/C * D
= 2.0 * 3.0/2/4.0 * 5.0
= 6.0/2/4.0 * 5.0
= 3.0/4.0 * 5.0
= 0.75 * 5.0
= 3.75

Gelijke prioriteiten van * en / dus van links naar rechts uitwerken

F.4 **1** 4 AC moet zijn: 4 * A * C en
2 A moet zijn: 2 * A
2 Er staat:

$$-b + \frac{\sqrt{b^2-4ac}}{2} * a$$

3 Er staat:

$$\frac{-b + \sqrt{b^2-4ac}}{2} * a$$

G.2 **1** 100 moet zijn: 100 *
3 y = 100 (1 + x (1 + x (1 + x)))
= 100 (1 + x (1 + x + x^2))
= 100 (1 + x + x^2 + x^3)
4 y = x betekent: de nieuwe waarde van y *wordt* de waarde van x
dus:

y = y + 1 *wordt* y = x + 1
y = y · x *wordt* y = (x + 1) x
y = y + 1 *wordt* y = (x + 1) x + 1
y = y · x *wordt* y = ((x + 1) x + 1) x
y = y + 1 *wordt* y = ((x + 1)x + 1)x + 1
y = y * 100 *wordt* y = (((x + 1) x + 1) x + 1) 100
dus y = 100 (1 + x + x^2 + x^3)

H.1 Uitspraak a is goed:
De nieuwe waarde van I *wordt* de oude waarde van I + de waarde van J
Uitspraak b is goed:
De tekst — 9 AUG 1944 — wordt toegekend aan de geheugenruimte die voor de string-variabele S gereserveerd is

I.2 In a, b en d wordt bij de laatste printopdracht naar een nieuwe regel gesprongen.

J.2 Voer het programma uit:

10	I = 0
20	I = 0 + 1 = 1
30 → 10	I = 0
20	I = 0 + 1 = 1
enz.	

Hoofdstuk 4

A.3 Uitwerken levert:
a A > B OR C < D
2 > 3 OR 4 < 5
FALSE OR TRUE
TRUE
b A + B > C + D OR A/B = C/D
2 + 3 > 4 + 5 OR 2/3 = 4/5
5 > 9 OR 0,6 = 0,8
FALSE OR FALSE
FALSE

B.2 Als de conditie na IF, de waarde true heeft, wordt in het algemeen de sprong uitgevoerd.
1 en 4 zijn fout omdat zowel naar een voorgaand, als een verder liggend statement gesprongen kan worden.

C.4 GOTO . . . IF is geen BASIC statement.
1, 2 en 3 zijn correcte BASIC programma's.

D.1 a Test de programma's met A = 1 en B = 1.
10 IF 1 ⩾ 1 THEN 200
Er wordt naar regel 200 gesprongen.
20 IF 1 > 1 THEN IF 1 = 1 GOTO 200
Conditie na het IF statement is FALSE, er wordt doorgegaan naar de volgende regel, dus er wordt *niet* naar regel 200 gesprongen.
Regel 10 en 20 hebben niet hetzelfde effect.

b De relationele expressies A < 7 en A − 7 < 0 zijn gelijkwaardig.
De regels 10 en 20 hebben hetzelfde effect.

E.2 a Test met
A = 1 B = 1 → F OR T OR F → T
A = 1 B = 2 → F OR F OR T → T
A = 2 B = 1 → T OR F OR F → T
Er wordt *altijd* naar regel 100 gesprongen.

b Test met
A = 1 B = 1 → F AND F AND F → F
A = 1 B = 2 → T AND T AND F → F
A = 2 B = 1 → F AND T AND T → F
Het resultaat is altijd F, er wordt *nooit* naar regel 100 gesprongen.

F.1 a Als in de lus een test op de waarde van een variabele opgenomen is, kan uit de programmalus gesprongen worden als deze variabele een grenswaarde overschrijdt.

b **Zie** § 4.5.

G.2 If (te testen conditie)

THEN ↗ – BASIC STATEMENT
     ↘ – REGELNUMMER

Als de conditie TRUE is wordt doorgegaan met de rest van het IF statement.

H.2 10 A krijgt de waarde 1, B de waarde 10
20 A krijgt de waarde 11
30 A en B zijn beide 11
40 A is kleiner dan 50 (11 < 50) → sprong naar regel 30
Regel 30 en 40 vormen een oneindige lus.

I.2 De waarden van S en T zijn achtereenvolgens:

	S	T
regel 10	1	
regel 30		1
regel 50	2	
regel 30		4
regel 50	6	
regel 30		(36) ← eindwaarde
regel 50	(42) ←	

Hierna volgt een sprong naar regel 20, omdat S > 20 (42 > 20) wordt naar regel 70 gesprongen. Het programma stopt.

J.1 De waarden van A, B en C zijn achtereenvolgens:

	A	B	C	
regel 10	1	4	1	
regel 20	5		2	1^e keer
regel 20	9		3	2^e keer
regel 20	13		4	3^e keer
regel 20	17		5	4^e keer

regel 30 A ⩾ 16 (17 ⩾ 16) → sprong naar regel 50
De procedure (regel 20, 30 en 40) wordt 4 keer doorlopen.

Hoofdstuk 5

A.4 Uitspraak a is fout:
Dit kan ook met de IF. THEN en de GOTO statements.
Zie § 5.1 en Afb. 5.15.
Uitspraak b is fout: Zie § 5.1.

B.4 Uitspraak a is fout:
Na de lus kan nog een stuk programma komen.
Uitspraak b is fout:
Als in de lus de IF... test staat kun je uit de lus springen.
Zie § 5.2 regel 1 t/m 6.

C.3 Commentaar statements beïnvloeden het programma niet.
Uitspraak b is fout:
Bijvoorbeeld als in afb. 5.9 het programmablok 'leeg' is hebben we een wachtlus.

D.3 **1** Is goed; de stapgrootte mag een negatief- en niet geheel getal zijn.
Zie blz. 84 de algemene vorm van het FOR statement.
2 Is goed; Q, R en R zijn numerieke variabelen.
3 Is fout; achter TO mag geen logische expressie staan.
4 Is goed; de begin- en eindwaarde en de stapgrootte mogen rekenkundige expressies zijn.

E. P en J doorlopen de volgende waarden:

	P	J	
	1	1	
	1	3	J loop (regel
	1	5	20, 30, 40)
	:	:	
P wordt	3	3	
2 opgehoogd	3	5	
	:	7	
	:	:	
	5	5	
	5	7	
	:	9	

Let op:
① Er wordt gevraagd naar de doorlopen waarden van P en J en niet naar de geprinte waarden.
② J> = P

F.3 **1** Levert 19 / 19 / 19 — want in regel 20 staat geen komma achter PRINT 19

2 Levert 19
Door 'LET I = 19' wordt meteen I>3, de lus wordt dus maar één maal doorlopen.

3 Levert 19 19 19
De lus wordt drie maal doorlopen. In elke doorgang wordt het getal 19 afgedrukt.
Regel 30 is een wachtlus.

4 De te printen waarde van I+J wordt als volgt verkregen:

I	J	I + J
7	12	19
7	10	17
7	8	15
9	12	21
9	10	19
9	8	17
11	12	23
11	10	21
11	8	19

G.4 **1** 5 × 25 × 10 = 1250
denkfout: regel 15 en 25 vergeten

2 5 × 25 × 10 + 2 = 1252
denkfout: regel 15 en 25 zitten in lussen

3 5 (25 (10 + 1)) = 1375
denkfout: regel 25 vergeten

4 5 (25 (10 + 1) + 1) = 1380

H.2

X	Y	Z	
1	1	1	
		2	
		3	3 regels
		4	
		5	
1	2	1	
		2	
		3	
		4	2 regels
		5	
1	3	1	
		2	
		3	
		4	
		5	1 regel
1	4	1	
		2	
		3	
		4	
		5	geen regel
1	5	1	
		⋮	
2	1	1	
		⋮	
		4	2 regels
		5	
2	2	5	1 regel
⋮	⋮	⋮	
3	1	5	1 regel
⋮	⋮	⋮	geen regels
			10 regels

I.2

J.4

I	J	K	(I+J+K)		aantal regels
1	1	1	3	3	1
2	1	1	4		
2	2	1	5	15	3
2	2	2	6		
3	1	1	5		
3	2	1	6		
3	2	2	7		
3	3	1	7	42	6
3	3	2	8		
3	3	3	9		
4	1	1	6		
4	2	1	7		
4	2	2	8		
4	3	1	8		
4	3	2	9		
4	3	3	10	90	10
4	4	1	9		
4	4	2	10		
4	4	3	11		
4	4	4	12		
5	1	1	7		
5	2	1	8		
5	2	2	9		
5	3	1	9		
5	3	2	10		
5	3	3	11		
5	4	1	10		
5	4	2	11	165	15
5	4	3	12		
5	4	4	13		
5	5	1	11		
5	5	2	12		
5	5	3	13		
5	5	4	14		
5	5	5	15	+	+
				S = 315	35 regels worden afgedrukt

S = S + I + J + K = 315

Hoofdstuk 6

A.1 Uitspraak a is goed; eenzelfde deelprogramma wordt niet steeds opnieuw uitgeschreven in hetzelfde programma.
De programma-opbouw wordt hierdoor overzichtelijker.
Uitspraak b is goed; als een zich herhalende opdracht als subroutine gedeclareerd wordt, dan hoeft deze maar éénmaal geschreven te worden.

B.2 Uit § 6.1 en afb. 6.1 blijkt dat antwoord 2 de juiste oplossing is.

C.1 **1** § 6.1 laat zien dat tussen ON en GOTO wel een rekenkundige maar geen relationele expressie mag staan, en K > = 3 is een relationele expressie (zie § 4.2: > = is een relationele operator! dus A > = B is een relationele expressie).
2 De reeks met regelnummers mag *wel* meer keren hetzelfde regelnummer bevatten.
3 Het integerdeel van K wordt altijd genomen, zie § 6.1.
4 Er mogen wel meer dan drie regelnummers achter GOTO staan.

D.3 **1** Als A < 0 dan K = 1 d.w.z. na GOTO wordt alleen maar naar regel 100 gegaan.
2 Het programma werkt voor alle waarden van A.
3 Als A = 0 dan wordt afgedrukt (GOTO 200) SIGN = 0
Als A > 0 dan wordt afgedrukt (GOTO 300) SIGN = 1
Als A < 0 dan wordt afgedrukt (GOTO 100) SIGN = −1
4 Men hoeft niet naar de inputgegevens terug te springen. Het programma kan ook opnieuw worden gestart.

E.3 Uitspraak a is goed; denk aan het *nesten* van subroutines.

```
—— SUBROUTINE 1
   SUBROUTINE 2
   RETURN
—— RETURN
```

Als RETURN eerder in 't programma geplaatst wordt dan de subroutine, dan weet de computer niet waarheen hij terug moet gaan.
Uitspraak b is fout; bij een multiple branch in een subroutine kunnen er verscheidene RETURN-statements zijn, waarvan er slechts één wordt uitgevoerd.

F.3 **1** D.m.v. GOTO kan men ook terugspringen in het programma waardoor men een lus krijgt. (Vb. zie onder 4.)
2 Het GOSUB-statement hoeft niet sneller te zijn dan het GOTO-statement.
3 In BASIC mogen we alléén naar een subroutine d.m.v. de aanroep GOSUB.
4 Zie § 6.3. Bij GOTO kunnen we naar een subroutine springen als we het regelnummer opgeven waar deze subroutine staat, *maar* als we bij RETURN aankomen wordt er naar een willekeurig regelnummer teruggekeerd dat toevallig in de stack staat.
vb. ad 1)

```
10 INPUT A
20 B = B + 1
30 IF B > = 1000  THEN 50
40 GOTO 20
50 PRINT A
60 END
```

G.4 De subroutine op regel 100 wordt drie keer aangeroepen, waarbij regel 30 en 40 ook terugkeeradressen zijn.
A wordt achtereenvolgens:
A = 1 GOTO 100
A = 10 RETURN naar 30 en GOTO 100
A = 100 RETURN naar 40 en GOTO 100
A = 1000 RETURN naar 50 en PRINT A

H.1 Voer het programma uit:
A = 50 B = 100
A = 50 + 100 = 150
A = 150/5 = 30
A = 30/5 = 6
PRINT levert: 6 100

I.3 Uitspraak a is goed; men bepaalt zelf, door het regelnummer achter GOSUB, naar welke subroutine men gaat.
Uitspraak b is fout; alleen de integerwaarde van de variabele wordt bekeken.

J.3 Bij invoer van −4 drukt de Teleac/DAI computer niets af want dan: GOTO 10, er wordt nieuwe input gevraagd, d.i. 4.2; de computer gaat dan naar regel 30 (regel 20 wordt gelezen tot THEN, er wordt niet aan de voorwaarden voldaan dus wordt er verder gegaan op regel 30). Daar wordt de integerwaarde van 4.2 genomen en GOSUB 400 uitgevoerd. Hier krijgt N$ een waarde 'VIER'.
Door de RETURN keren we terug naar regel 40 waar N$ geprint wordt.

Hoofdstuk 7

A.3 Het DATA statement zet gegevens in het geheugen, het READ statement leest deze gegevens uit het geheugen en kent ze toe aan variabelen.

B.1 **a** De uitspraak is goed bijv.:
READ A
READ B
DATA 1, 2
b Deze uitspraak is ook goed bijv.:
DATA 5
DATA 6
READ C

C.3 DATA 2 * 2 is niet toegestaan.
Zie onder 'Let wel!', blz. 80 en het daarop volgende voorbeeld.

D.4 De volgorde van READ en DATA statements is onbelangrijk.

E.3 **a** Deze uitspraak is goed, bijvoorbeeld:
READ A, B, C
b Deze uitspraak is fout.
Toegestaan is:
DATA 3, 5, 8
READ A
READ A
De variabele A heeft nu de waarde van het tweede ingelezen getal gekregen, dus A = 5.

F.4 Het RESTORE statement zorgt ervoor dat de elementen uit het eerstvolgende READ statement gelezen worden vanaf het eerste element uit de DATA-lijst.

G.4 Het DIM statement reserveert geheugenruimte voor een rij getallen, een array.

H.4 **a** De uitspraak is fout.
DIM A (83) declareert een twee-dimensionaal array.
Het aantal subscripts in een DIM-statement geeft het aantal dimensies van het array.
De waarde van een subscript is een maat voor de grootte van de bijbehorende dimensie, nl. grootte van de dimensie = waarde subscript + 1.
LET A (3) = 5 is niet correct.
Het aantal subscripts moet gelijk aan 2 zijn.
LET A (3, 2) = 5 is wel correct.
b De uitspraak is fout. Een subscript mag niet negatief zijn.
DIM B (2, 4, 6)
LET B (1, 3, −1) = 2 is niet toegestaan

I.1 Beide uitspraken zijn goed
a DIM A (8) : DIM B (6, 2) is correct.
b DIM C (2, 6 * 4) is toegestaan.
Zie de algemene vorm van een DIM statement op blz. 115.

J.2 **a** De uitspraak is fout.
DIM A (10, 8) declareert een Array A van 11 × 9 = 99 elementen.
b De waarden van de subscripts in het tweedimensionale array A (10, 8) lopen voor de eerste subscript van 0 t/m 10 en voor de tweede van 0 t/m 8.

We hebben dus onderstaande mogelijkheden:

(0,0)	(0,1)	(0,2)	(0,3)	(0,4)
(1,0)	(1,1)	(1,2)	(1,3)	(1,4)
(2,0)	(2,1)	(2,2)	(2,3)	(2,4)
(3,0)	(3,1)	(3,2)	(3,3)	(3,4)
(4,0)	(4,1)	(4,2)	(4,3)	(4,4)
(5,0)	(5,1)	(5,2)	(5,3)	(5,4)
(6,0)	(6,1)	(6,2)	(6,3)	(6,4)
(7,0)	(7,1)	(7,2)	(7,3)	(7,4)
(8,0)	(8,1)	(8,2)	(8,3)	(8,4)
(9,0)	(9,1)	(9,2)	(9,3)	(9,4)
(10,0)	(10,1)	(10,2)	(10,3)	(10,4)

(0,5)	(0,6)	(0,7)	(0,8)
(1,5)	(1,6)	(1,7)	(1,8)
(2,5)	(2,6)	(2,7)	(2,8)
(3,5)	(3,6)	(3,7)	(3,8)
(4,5)	(4,6)	(4,7)	(4,8)
(5,5)	(5,6)	(5,7)	(5,8)
(6,5)	(6,6)	(6,7)	(6,8)
(7,5)	(7,6)	(7,7)	(7,8)
(8,5)	(8,6)	(8,7)	(8,8)
(9,5)	(9,6)	(9,7)	(9,8)
(10,5)	(10,6)	(10,7)	(10,8)

Aan elk element van het array kan een waarde toegekend worden, bijv. A (3,4) = −17.
Het hoogste subscript is dus 10.
Het laagste subscript is dus 0.

Hoofdstuk 8

A.3 Een BASICfunctie is een vast in de machine geprogrammeerde routine. Zie hoofdstuk 8 onder: Het begrip 'functie'.

B.1 Met de terugkeer via RETURN hebben we niets te maken, functies worden gebruikt d.m.v. een aanroep onder naam. Zie onder: 'Resultaat'.

C.1 Uitspraak a is goed. Zie onder voorbeeldprogramma.
Uitspraak b is goed. De waarde van een functie kan weer als argument voor een functie gebruikt worden. Zie onder: 'Resultaat'.

D.4 Uitspraak a is fout. Bijvoorbeeld om de functie ABS (X) te bepalen hebben we alle cijfers van het getal X nodig.
Uitspraak b is fout SGN (0) = 0.

E.4 Uitspraak a is fout. Neem X = −1.1

$$SGN(-1.1) * INT(ABS(-1.1)) \stackrel{?}{=} INT(-1.1)$$
$$-1 \quad * \quad 1 \quad \neq -2$$

Uitspraak b is fout. Neem X = −1.1

$$FRAC(-1.1) \stackrel{?}{=} -1.1 - INT(-1.1)$$
$$-0.1 \stackrel{?}{=} -1.1 - (-2)$$
$$-0.1 \neq 0.9$$

F.4 Uitspraak a is fout, want achter RND moet een argument staan.
Uitspraak b is fout; bij aanroep van RND (X) produceert de randomgenerator een willekeurig gebroken getal tussen 0 en X.

G.2 Uitspraak a is fout; ABS (0) = 0, nul is nòch positief nòch negatief.
Uitspraak b is goed; zie voorbeeld op blz. 141.

H.3 Uitspraak a is goed; X = 10↑Y
LOGT (X) = LOGT (10↑Y) = Y want per definitie is Y = LOGT (10↑Y).
Uitspraak b is fout; zie a en X = 10↑Y is ongelijkwaardig aan X = EXP (Y).

I.1 Uitspraak a is goed; zie tekening. COS (H) = B/C $\stackrel{?}{=}$ COS (ACOS (B/C)). Dit klopt want COS en ACOS heffen elkaar op.
Uitspraak b is goed, want LOG (EXP (X)) = X, want LOG en EXP heffen elkaar op.

J.3 Uitspraak a is goed; ABS (0) = 0; INT (0.1) = 0; FRAC (1) = 0.
Uitspraak b is fout;
SQR (0) = 0
COS (π/2) = 0
RND (X) $\neq$ 0 dit levert nl. als resultaat een getal tussen 0 en X.

Hoofdstuk 9

A.3 Uitspraak a is goed, zie toestandsdiagram e.v.
Uitspraak b is fout; vooronderzoek houdt onder andere in: Kan het probleem met de computer worden opgelost.

B.3 De begrijpelijkheid van het programma is niet van belang voor de werking ervan.

C.2 Zie tekst.

D.3 Uitspraak a is goed;
Uitspraak b is fout; GOTO of GOSUB is juist een onvoorwaardelijke sprong.

E.2 Uitspraak a is fout; een subroutine wordt met GOSUB aangeroepen.
Uitspraak b is goed; wanneer het aantal keren dat een opdracht moet worden herhaald, van te voren kan worden berekend, is het volgende BASIC statement bruibaar: FOR... TO. Zoals in hoofdstuk 5 reeds is vermeld, moet zo'n loop worden afgesloten met een NEXT-statement.

F.3 Uitspraak a is goed; overdracht in BASIC vindt plaats via vast afgesproken geheugenplaatsen.
Uitspraak b is fout; Top-to-down aanpak heeft te maken met een bepaalde methode van probleemoplossing.

G.3 Als de doorzichtigheid niet in eerste instantie de meeste aandacht heeft gekregen, kan men veel moeilijkheden krijgen als men 1) besparing van geheugenruimte, 2) verhoging van executiesnelheid en 3) gebruik van REM-statements wil toepassen.

H.1 Uitspraak a is goed;
Uitspraak b is goed;

I.2 Uitspraak a is fout; men ziet anders door de bomen het bos niet meer; zie par. 9.6.
Uitspraak b is goed; zie par. 9.7, 9.8, basisvorm 4.

J.4 Uitspraak a is fout; zie par. 9.7.
Uitspraak b is fout; zie par. 9.8. De afwezigheid van fouten tijdens het testen is nog geen bewijs dat het programma correct is, alhoewel er zulke eenvoudige programma's te bedenken zijn die wel uitputtend getest kunnen worden op hun correctheid.

Hoofdstuk 10

A.2 Uitspraak a is fout; een string kan ook zonder aanhalingstekens worden ingevoerd, alleen mag de string dan niet met een cijfer beginnen en worden voorafgaande spaties niet gezien.
Uitspraak b is goed.

B.1 3 * 4 * 255 = 3060 tekens.

C.3 Uitspraak a is goed; zie hoofdstuk par. 10.2 onder programmavoorbeeld.
Uitspraak b is fout; met de subscripts kunnen we een bepaalde string uit het array selecteren.

D.2 Uitspraak a is fout; het geeft de decimale waarde van de ASCII code van het eerste teken X$.
Uitspraak b is goed; zie par. 10.5.

E.1 Uitspraak a is goed; zie par. 10.4 onder het 1[e] voorbeeld.
Uitspraak b is goed; zie par. 10.5.

F.2 Uitspraak a is fout; A$ kan wel cijfersymbolen maar geen getalwaarden bevatten.
Uitspraak b is goed; in PRINT kunnen zowel getallen als strings worden geprint.

G.3 Uitspraak a is goed; zie par. 10.2.
Uitspraak b is fout; zie par. 10.2.

H.4 Uitspraak a is fout; de kortste string wordt uitgebreid gedacht met blanco's en elk volgend teken van de ene string wordt steeds vergeleken met het volgende teken van de andere string tot er een verschil is bijvoorbeeld ASC("U") > ASC("AA"). Zie ook het tweede voorbeeld.
Uitspraak b is fout; logische operaties op strings zijn niet toegestaan; relationele operaties op strings zijn wel toegestaan; het plus-teken levert een samenvoeging van strings (concatenatie).

I.1 Uitspraak a is goed; A(2,2) is een getallenarray bestaande uit drie rijen en drie kolommen.
De array kan maximaal bestaan uit 3 * 3 = 9 getallen.
Zie ook de motivatie van vraag J van les 7.
Uitspraak b is goed; A$(2,2) is een string-array. De string-array kan maximaal uit negen strings bestaan.

J.2 Uitspraak a is fout; er mogen geen variabelen in het data statement staan, echter wel numerieke waarden voor variabelen.
Uitspraak b is goed; zie par. 10.2.

Hoofdstuk 11

A.3 Uitspraak a is goed; OORSPRONG + STUIT krijgt bij elke lusdoorgang steeds dezelfde constante waarde.
Uitspraak b is fout. Dit is geen BASIC statement. De berekende waarde wordt niet aan een variabele toegekend.

B.1 Uitspraak a is goed; De FOR. . .NEXT loop wordt vervangen door een teller.
Regel 110 t/m 130 worden even vaak uitgevoerd.
Uitspraak b is goed. In deze regels wordt de horizontale rij punten geprint.

C.1 Uitspraak a is goed; zie hiertoe de tekst op blz. 86 en blz. 132.
Uitspraak b is goed.
150 FOR K = 1 TO KEER
en 150 FOR K = 0 TO KEER−1
hebben hetzelfde effect. De lus wordt even vaak doorlopen.

D.2 Uitspraak a is fout. Het resultaat van regel 30 is een geheel getal.
Uitspraak b is goed. RAAD is al een positief geheel getal of nul, dus ABS (INT(RAAD)) levert een positief geheel getal of nul.

E.3 Uitspraak a is goed. Zie hoofdstuk 10.
Uitspraak b is fout. Vergelijk bijvoorbeeld het getal 5.3 met het getal 5.8.
Als ze beide op 5 afgerond zouden worden is er geen onderscheid meer.

F.2 Uitspraak a is fout. Januari is maand 1.
Uitspraak b is goed. Zie ook regel 40 van het programma.

G.1 Uitspraak a is goed. Men mag meer dan één statement op een regel plaatsen, mits ze gescheiden zijn door een dubbele punt.
Uitspraak b is goed. Het alternatieve programma mist b.v. de statements op regelnr. 160 t/m 280 en 780.

H.4 Aantal schijven vergroten:
- dus de dimensie van de schijf veranderen (regel 40)
- regel 300 geeft het aantal schijven aan
- regel 980, 990 en 1000 bevatten data van de schijven.

I.4 Uitspraak a is fout. Het resultaat van regel 650 is dan een afronding op halve guldens.
Uitspraak b is fout.
Regel 630 SCHULD = SCHULD − AFLOS blijft hetzelfde voor elke hypotheekvorm.

J.4 Regel 300 moet veranderen, want K = O TO 9 moet worden K = O TO 8.
De regels 980, 990 en 1000 bevatten de data en moeten dus ook veranderen.

Hoofdstuk 12

A.4 Uitspraak a is fout. Het grootste deel van de interpreter staat in ROM geheugen, maar er wordt ook gebruik gemaakt van RAM geheugen.
Uitspraak b is fout. In de memory-map (indeling van het geheugen) wordt van het geheugen aangegeven welke soort het is (RAM, (P)ROM, memory-mapped I/O).
Zie fig. 12.2.

B.2 In BASIC roepen we een in machinetaal geschreven programma aan met het statement CALLM "adres". BASIC behandelt dat programma als een subroutine en om alles goed te laten verlopen moet dit programma afgesloten zijn met een RETURN instructie om weer naar het juiste regelnummer terug te keren voor een correct verloop van de rest van het BASIC programma.

C.4 Uitspraak a is fout. Op het 8 bits getal in de Accu wordt eventueel de correctie uitgevoerd naar een decimaal getal van 2 × 4 bits.
Zie fig. 12.6.
Uitspraak b is fout. Deze instructie telt nl. ook de carry erbij op. Zie de 8080 instructieset.

D.4 Zie de 8080 instructieset.

E.1 Zie de 8080 instructieset in de kolom "NO of States".

F.3 Register B heeft als codering 000
MOV M,B heeft als OP CODE
0111 0000
d.i. 7 0
dus 70 hexadecimaal.

G.3 Uitspraak a is goed. Zie kolom "condition flags".
Uitspraak b is fout. De carryflag *wordt* nul.

H.2 1111 0001
1111 0001 EXCR
____
0000 0000
EX OR als twee bits gelijk zijn is het resultaat 0
als twee bits ongelijk zijn is het resultaat 1

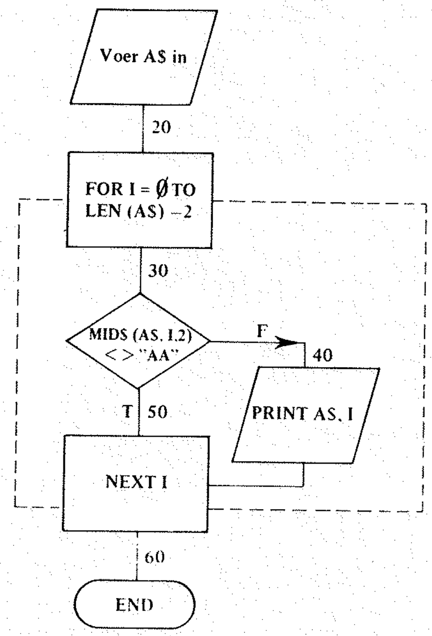
I.1 Uitspraak a is goed. Dit is nl. het doel van een compiler.
Tijdens de executie hoeft niet steeds weer regel voor regel vertaald te worden.
Uitspraak b is goed. De compiler is niet meer actief tijdens de executiefase.

J.3 Zie de 8080 instructieset in de Branch Group.

Hoofdstuk 13

A.1 Uitspraak a is goed.
3 * INT (A/3) = A is gelijkwaardig met
INT (A/3) = A/3 is gelijkwaardig met
A is een geheel 3-voud.
Uitspraak b is goed.
ASC(CHR$(Q)) = Q dit geldt voor alle toegestane waarden van Q.
Zie afb. 10.7.
Opmerking: CHR$(ASC(Q$)) ≠ Q$ neem bijvoorbeeld Q$ = "A B", dit levert A op.

B.4 Uitspraak a is fout. Als P positief of nul is wordt de eerste regel 10 uitgevoerd door Q = P te maken. De tweede regel 10 wordt voor dit geval overgeslagen. Uitspraak b is fout. INT(2.3) + 0.5 = 2.5 ≠ 2 Correct zou zijn INT (A + 0.5).



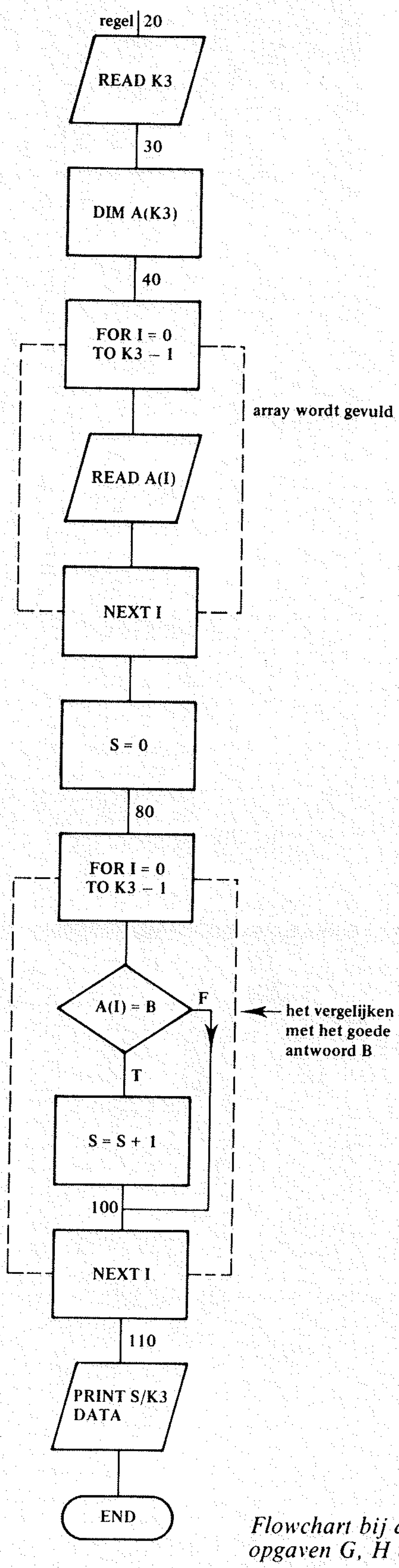
Flowchart bij de opgaven C, D, E en F

C.2 PRINT I geeft de plaats aan.
MID$ (A$, 1,2) geeft 2 tekens uit A$, beginnend op positie I.
Opmerking: het meest linkse symbool van A$ heeft positie nul.

D.3 D1 en D2 zijn fout omdat I niet verhoogd wordt; het programma blijft lussen vanaf de eerste keer dat AA gevonden is.
D3 is goed. NEXT I zorgt voor de ophoging van I i.v.m. de volgende plaats.
D4 is fout; nadat I de eerste keer geprint is, stopt het programma.

E.4 Uitspraak a is fout. De geprinte waarde van I geeft wel degelijk de plaats van het cijferpaar aan. Uitspraak b is fout. We krijgen geen foutmelding; er wordt aan de conditie voldaan omdat we 2 letters met 3 letters vergelijken. Vervangen we het cijfer 2 door 3 in regel 20 en 30 dan kan er wel geprint worden.

F.1 We krijgen als foutmelding: NUMBER OUT OF RANGE IN LINE 30. I begint met 1. LEFT$ (A$, I) stelt dan de meest linkse letter van A$ voor. Omdat hierna de 2 meest rechtse letters van deze linkse letter worden gevraagd krijgen we bovenstaande foutmelding.

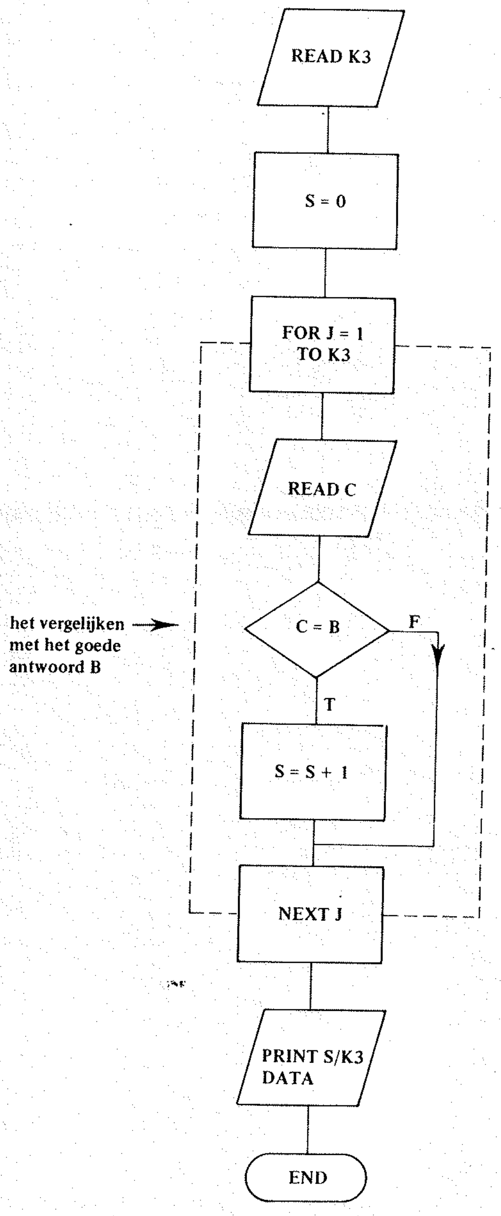


Flowchart bij de opgaven G, H en I

G.2 De dimensie is 1 te hoog genomen, er wordt immers van nul af geteld.
DIM A(K3 – 1) heeft K3 plaatsen.

H.4 Uitspraak a is fout. We krijgen dan de foutmelding: OUT OF DATA IN LINE 50.
Uitspraak b is fout. Hier wordt dan het totale aantal goede antwoorden geteld en niet de gemiddelde score.

I.3 Uitspraak a is goed.
Uitspraak b is fout. Regel 20 mag niet vervallen. K3 stelt het aantal kandidaten voor. Regel 30 mag vervallen.



J.2 0 < RND (1) < 1
0 < 49 * RND (1) < 48
0 = < INT (49 * RND (1)) = < 48
1 = < INT (49 * RND (1)) + 1 = < 49
Opmerking: A (0) = 0
Regel 60 geeft aan dat er bij één en zelfde I net zolang wordt teruggesprongen totdat er een geheel nieuwe K waarde wordt gevonden; dit programma stelt een loting zonder teruglegging voor.

Hoofdstuk 14

A.3 We hebben hier te maken met een memory-mapped I/O. Uit de instructieset in hoofdstuk 12 volgt dat: LXI H,OFOOOH 10 states en
MOV M,C 7 states heeft

Samen 17 states

B.4 Bij memory-mapped I/O worden de I/O poorten als het ware op geheugenplaatsen gezet, alle 16 bits kunnen voor I/O poorten worden gebruikt.
Zie bijvoorbeeld bij antwoord A3: adres FOOOH.

C.3 64K – 62K = 2K I/O poorten.
Bij eenzelfde adres hoort één I en één O poort, die respectievelijk afzonderlijk door de besturingssignalen "lees geheugen" en "schrijf geheugen" kunnen worden gekozen.

D.4 Uitspraak a is fout. IN en OUT instructies worden voor I/O mapped-I/O gebruikt.
READ en WRITE besturingssignalen worden voor memory-mapped I/O gebruikt.
Beide kunnen binnen één systeem optreden.
Uitspraak b is fout. STA en LDA mogen worden gebruikt.

E.3 De geheugenpointer (H,L) wordt immediate met FFFFH gevuld.
XRA A maakt de inhoud van A nul.
ADD M de inhoud van adres FFFFH wordt opgeteld bij A = O en komt in A.
De parity flag wordt aangepast aan de input data (zie de 8080 instructieset).

F.2 De reeks instructies bepalen d.m.v. het aantal states de tijd per datawoord.
We kunnen dus speken van een bepaalde snelheid (zie ook antwoord A.3).

G.3 I/O mapped-I/O (i.v.m. input instructie) gebruikt altijd de accumulator.
Zie de Stack, I/O, and machine control group van de instructieset.

H.2 De output latch houdt de data langdurig vast. Een nieuwe output instructie overschrijft de latch inhoud.
Een input instructie heeft op de latch geen invloed.

I.2 Uitspraak a is fout. Direct transport is mogelijk; zie hierover het eerste voorbeeld van hoofdstuk 13 par. 3.2.
Uitspraak b is goed. Data kan bijvoorbeeld verloren gaan als de status niet in orde is.

J.4 De oplossingen 1, 2 en 3 zijn fout, er vindt nergens datatransport plaats naar output poort 1000H.

Hoofdstuk 15

A.1 Zie hoofdstuk 14 par. 3 en afb. 14.6.
De randapparatuur geeft eerst de data en dan de status.

B.1 110 bits per seconde = 110 bits per 1000 milliseconden = 1 bit per 9,1 milliseconden.

C.2 Door het startbit bij serie I/O weet de processor wanneer een nieuw woord begint.
Hij kan de woordsynchronisatie starten en er is onderscheid met de rusttoestand.
(laag-hoog signaal).

D.1 Uitspraak a is goed. Een woord wordt met start- en stopbits in zijn geheel overgezonden.
Uitspraak b is goed. Veel software is al in de USART ingebouwd.

E.1 Uitspraak a is goed. Zie hoofdstuk 14 par. 6.
Uitspraak b is goed. Zie hoofdstuk 14 par. 6.

F.1 Uitspraak a is goed. Zie hoofdstuk 15 par. 3.
Uitspraak b is goed. Zie hoofdstuk 15 par. 3. Input-data worden pas dan gelezen als het signaal stabiel is, dan wordt input-data accepted hoog en de begeleidende puls ook hoog.

G.1 Uitspraak a is goed. Zie hoofdstuk 15 par. 2. Dit is het principe van hand-shake, de poort is geblokkeerd voor nieuwe data totdat de hand-shake flipflop gereset is.
Uitspraak b is goed. Zie hoofdstuk 15 par. 3.

H.2 Uitspraak a is fout. Bij gebruik van JM en het controlesignaal bit 7 = 0 zal de data in poort FF gelezen worden; dit is niet de bedoeling.
Uitspraak b is goed. Als A7 één wordt ten gevolge van IN OFEH dan wordt CY gelijk aan 1 ten gevolge van RAL ((CY)←(A7)) en dan wordt de sprong tengevolge van de instructie JNC dus overgeslagen.

I.1 Uitspraak a is goed. Zie hoofdstuk 15 par. 4.
Uitspraak b is goed. Zie hoofdstuk 15 par. 4.

J.1 Zie hoofdstuk 15 is het einde van par. 4. Afb. 15.11. Eerst de pariteitsbit uitsturen, dan de data samen met het output-data available signaal (op een andere outputpoort) geven.

Hoofdstuk 16

A.4 Met behulp van het monitorprogramma kan men op machineniveau geheugenplaatsen uitlezen; de inhoud van geheugenplaatsen veranderen, enz.
Zie verder de inleiding van hoofdstuk 16.

B.4 Uitspraak a is fout. Zie hoofdstuk 16 par. 2. De programmeur moet exact weten hoe de structuur van de machine is om een machine correct te kunnen programmeren.
Uitspraak b is fout.

C.1 Met het display commando maakt men alleen de inhoud van een gedeelte van het geheugen op het scherm zichtbaar. Met het display commando kan men geen output-data in poorten schrijven.

D.3
- M.b.v. het S commando kan men een reeks geheugenplaatsen vanaf het opgegeven adres veranderen. Zie afb. 16.8.
- M.b.v. het M commando kan men een aaneengesloten geheugenblok, dat loopt van het laag adres tot het hoog adres, verplaatsen naar een ander geheugenblok, dat begint bij het destination adres. Zie afb. 16.9 en 16.10.
- M.b.v. het X commando zal de machine de inhoud van alle registers; de programmateller; de stackpointer en de waarde van de speciale registers laten zien. Zie afb. 16.11.
- M.b.v. het F commando kan men een bepaald aaneengesloten geheugenblok vullen met een opgegeven waarde (byte). Zie afb. 16.6 en 16.7.

F.3 Uitspraak a is goed. Programma-ontwikkeling zo mogelijk eerst achter het bureau.
Uitspraak b is fout. Zie hoofdstuk 16 par. 4.1. Monitorcommando's.

G.3 Uitspraak a is goed. De inhoud van geheugenplaatsen kan met het monitorprogramma worden veranderd.
Uitspraak b is fout. Zie hoofdstuk 16 par. 1. We moeten zelf de adressen, die in het programma worden gebruikt, berekenen.

H.2
1 = fout want dan blijft de bit op 1 staan.
2 = goed want het aantal keren dat de lus wordt doorlopen is een maat voor de pulsduur.
3 = fout want bit 3 van het A register wordt 1; er treedt wel dezelfde puls op zoals eerst op bit 0.
4 = fout want bit 0 blijft 1.

I.2 Zie hoofdstuk 16 par. 4.3. M.b.v. het LOOK commando kunnen we een programma opstarten *en* tijdens de executie tonen wat er zoal gebeurt met de registers.

J.3

codering A	= 111			
MVI r	= 00DD	D 110	met r = A	
dus MVI A	= 0011	1 110		
instructiecode	= 3	E		

Hoofdstuk 17

A.4 De input-functie heeft als argument het adres van de inputpoort.
INP(I) haalt de waarde uit inputpoort I op.

B.3 De I/O poorten worden bij memory-mapped I/O als deel van het geheugen beschouwd.
Er zijn geen afwijkende instructies voor I/O. Dezelfde instructies die op het geheugen betrekking hebben, hebben ook betrekking op de I/O poorten. Zie fig. 17.5.

C.1 De waarde, die geschreven moet worden, ligt tussen 0 en 255 en het adresbereik is van 0 tot en met 65535.
POKE I,J = POKE ADRES, WAARDE.

D.4 Uitspraak a is fout. INP(X) en OUT(I,J) zijn zonder uitbreidingen (D.A.I. REAL WORLD CARDS) niet te gebruiken.
Uitspraak b is fout. De methoden om I/O via INP en OUT instructies te verwerken wordt directe I/O genoemd.

E.4 Uitspraak a is fout. Met WAIT wordt gewacht op een bepaald signaal uit de buitenwereld.
Uitspraak b is fout. WAIT TIME wordt gebruikt bij het maken van wachtlussen.
Zie het programma "Een tijdje wachten".

F.2 Uitspraak a is fout. WAIT MEM I,J[,K] voor memory-mapped I/O.
WAIT I,J[,K] voor I/O poorten van de Real World Card.
Uitspraak b is goed. K is de omkeerwaarde die een Exclusive OR vormt met de binnengekomen waarde. K mag ook weggelaten worden wat wil zeggen dat de Exclusive OR wordt weggelaten.

G.2 Zie de tekst van de WAIT-functie, dit wat betreft de AND bewerking.

H.4 VARPTR(B) heeft als waarde:
1) als B een variabele is, het geheugenadres van de variabele B.
2) als B een array is, het geheugenadres van het eerste element van array B.

I.3 Uitspraak a is goed. Het BASIC statement OUT moet worden omgezet in machinecode, dit vergt extra tijd.
Uitspraak b is fout.

J.3 Uitspraak a is goed. Met POKE I,J kan men informatie naar gespecificeerde geheugenplaatsen toeschrijven, dus ook naar plaatsen die al zinvolle inhoud hadden. De oorspronkelijke inhoud gaat dan verloren.
Uitspraak b is fout. INP(X) en OUT I,J hebben alleen te maken met I/O.

Appendix

Adressenbestandprogramma 'Claire'

Op de volgende pagina's vindt u de uitvoering van het adressenbestandprogramma "Claire" behorende bij hoofdstuk 16 en 17.
Programma is geschreven op de Apple II Personal Computer.
De Apple computerconfiguratie bestond uit 48K byte geheugen + floppy disk en de Apple-soft ROM kaart.
De floppy disk controller wordt gezet in slot 6.
De printer interface kaart hoort in slot 2, de gebruikte printer is de Anadex printer. (inlichtingen printer: Fa. Telerex te Best)

```
5  REM  CLAIRE 5 JANUARI 1980
10  HOME
20  DIM N$(100)
30  DIM A$(100)
40  DIM P$(100)
50  DIM W$(100)
60  DIM K$(100)
70  PRINT "*****************************************"
80  PRINT "                  C L A I R E"
90  PRINT "          EEN ADRESBESTAND PROGRAMMA"
100  PRINT
110  PRINT "*****************************************"
120  PRINT
130  PRINT "U KUNT KIEZEN UIT DE VOLGENDE"
140  PRINT "OPDRACHTEN:"
150  PRINT "INVOER(BESTAND MAKEN OF UITBREIDEN)"
160  PRINT "INHOUD(HET TONEN VAN HET BESTAND)"
170  PRINT "ZOEK(IN HET BESTAND)"
180  PRINT "SORTEER(GESPECIFICEERD PER REGEL)"
190  PRINT "VERANDER(IN HET BESTAND)"
200  PRINT "LABELS(PLAK-ETIKETTEN VAN HET BESTAND)"
210  PRINT "BEWAAR(OP SCHIJF OF CASSETTE)"
220  PRINT "EINDE(STOPPEN MET HET PROGRAMMA)"
230  PRINT "HELP(TOON DE OPDRACHTMOGELIJKHEDEN)"
240  PRINT
250 F1 = 0
260  INPUT "WAT IS UW OPDRACHT? : ";O$
270  PRINT
280  RESTORE
290  READ S$,T
300  IF S$ = "$$" THEN 430
310  IF  LEFT$ (S$,3) <  >  LEFT$ (O$,3) THEN 290
320  ON T GOSUB 1000,2000,3000,4000,5000,6000,7000,8000,9000
330  DATA "INVOER",1
340  DATA "INHOUD",2
350  DATA "ZOEK",3
360  DATA "SORTEER",4
370  DATA "VERANDER",5
380  DATA "LABELS",6
390  DATA "BEWAAR",7
```

```
400  DATA "EINDE",8
410  DATA "HELP",9
420  DATA "$$",10
430  PRINT "ONBEKENDE OPDRACHT...,NOG EENS INVOEREN"
440  GOTO 240
1000  HOME
1010  IF N > 0 GOTO 1400
1020  PRINT "**INVOER**"
1030  PRINT "=========="
1040  PRINT
1050  PRINT "GAAT U EEN NIEUW BESTAND OPBOUWEN"
1060  PRINT "OF WILT U EEN OUD BESTAND OPHALEN?"
1070  INPUT "NIEUW OF OUD BESTAND(N/O)?:  ";B$
1080  IF B$ = "O" THEN  GOTO 1600
1090 N = 1
```

```
1100  HOME
1110  PRINT
1120  PRINT "U KUNT NU ADRESSEN INVOEREN"
1130  PRINT "OM MET INVOER TE EINDIGEN:TIK-STOP-IN"
1140  PRINT
1150  FOR P = N TO 100
1160  PRINT "<";P;">"
1170  PRINT "====="
1180  INPUT "NAAM        :  ";N$(P)
1190  IF N$(P) = "STOP" GOTO 1300
1200  INPUT "ADRES       :  ";A$(P)
1210  IF A$(P) = "STOP" GOTO 1300
1220  INPUT "POSTCODE    :  ";P$(P)
1230  IF P$(P) = "STOP" GOTO 1300
1240  INPUT "WOONPLAATS  :  ";W$(P)
1250  IF W$(P) = "STOP" GOTO 1300
1260  INPUT "KENMERK     :  ";K$(P)
1270  IF K$(P) = "STOP" GOTO 1300
1280  PRINT
1290  NEXT P
1300  PRINT
1310  PRINT "EINDE INVOER"
1320 N = P - 1
1330  PRINT "ER ZIJN NU ";N;" ADRESSEN IN HET BESTAND"
1340  GOTO 240
1400  HOME
1410  PRINT "ER ZIT AL EEN BESTAND IN HET GEHEUGEN!!"
1420  INPUT "WILT U DAT OVERSCHRIJVEN????(J/N):  ";C$
1430  IF C$ = "N" GOTO 1460
1440 N = 1
1450  GOTO 1100
1460  PRINT
1470  INPUT "WILT U BESTAAND BESTAND UITBREIDEN?(J/N) :  ";U$
1480  IF U$ = "N" GOTO 240
1490 N = N + 1
1500  GOTO 1100
1595  REM  LAADT ROUTINE (SCHIJF)
1600  HOME
1610  PRINT "KLAAR OM ADRES-BESTAND VAN SCHIJF OP TE HALEN..."
1620 D$ =  CHR$ (4)
1630  PRINT
1640  PRINT "OP DE DISK STAAN DE VOLGENDE BESTANDEN:"
1650  PRINT D$;"CATALOG"
1660  PRINT
```

```
1670  PRINT "BESTANDEN WAABIJ EEN -T- VOORAAN STAAT"
1680  PRINT "KUNNEN INGELEZEN WORDEN"
1690  PRINT
1700  INPUT "WELK BESTAND KIEST U? : ";F$
1710  PRINT D$;"OPEN";F$
1720  PRINT D$;"READ";F$
1730  INPUT N
1740  FOR X = 1 TO N
1750  INPUT N$(X)
1760  INPUT A$(X)
1770  INPUT P$(X)
1780  INPUT W$(X)
1790  INPUT K$(X)
1800  NEXT X
1810  PRINT D$;"CLOSE";F$
1820  GOTO 240
```

```
2000  HOME
2010  PRINT "**INHOUD**"
2020  PRINT "=========="
2030  PRINT
2040  PRINT "WELKE ADRESSEN WILT U ZIEN?"
2050  PRINT "U KUNT KIEZEN UIT ALLES(A), EEN(E)"
2060  PRINT "OF EEN BEREIK(B)"
2070  PRINT
2080  INPUT "UW KEUZE? (A/E/B) : ";K$
2090  IF K$ <  > "A" THEN 2140
2100 T1 = 1
2110 T2 = N
2120  GOTO 2400
2140  IF K$ <  > "E" THEN 2190
2150  INPUT "WELK ADRESNUMMER? : ";T1
2160  IF T1 > N THEN 2310
2170 T2 = T1
2180  GOTO 2400
2190  IF K$ = "B" GOTO 2220
2200  PRINT "UW KEUZE IS NIET MOGELIJK...NOG EENS.."
2210  GOTO 2070
2220  INPUT "WAT IS HET EERSTE ADRESNR.?: ";T1
2230  IF T1 < 1 THEN T1 = 1
2240  IF T1 > N THEN 2310
2250  INPUT "WAT IS HET LAATSTE ADRESNR.?: ";T2
2260  IF T2 > N THEN 2310
2270  IF T2 =  > T1 THEN 2400
2280  PRINT "FOUT!! LAATSTE ADRES KLEINER DAN EERSTE ADRES.... NOG EENS..
    .."
2290  GOTO 2030
2310  PRINT "BESTAND HEEFT SLECHTS ";N;" ADRESSEN..."
2320  GOTO 2070
2400  HOME
2410  FOR W = T1 TO T2
2420  PRINT "<";W;">"
2430  PRINT "---"
2440  PRINT
2450  PRINT N$(W)
2460  PRINT A$(W)
2470  PRINT P$(W);"  ";W$(W)
2480  PRINT K$(W)
2490  PRINT
2500  NEXT W
2510  GOTO 240
```

```
3000  HOME
3010  PRINT "**ZOEK**"
3020  PRINT "========"
3030  PRINT
3040  PRINT "WAAR MOET NAAR GEZOCHT WORDEN, OFWEL WAT IS DE SLEUTEL?"
3050  PRINT
3060  INPUT "GEEF SLEUTELWOORD : ";Z$
3070 F =  LEN (Z$)
3080  FOR Z = 1 TO N
3090  IF  LEFT$ (N$(Z),F) = Z$ GOTO 3300
3100  IF  LEFT$ (A$(Z),F) = Z$ GOTO 3300
3110  IF  LEFT$ (P$(Z),F) = Z$ GOTO 3300
3120  IF  LEFT$ (W$(Z),F) = Z$ GOTO 3300
3130  IF  LEFT$ (K$(Z),F) = Z$ GOTO 3300
3140  NEXT Z
3150  IF F1 = 1 GOTO 240
3160  PRINT
3170  PRINT "SLEUTELWOORD KOMT NIET IN BESTAND VOOR.."
3180  INPUT "WILT U EEN NIEUWE SLEUTEL INVOEREN?(J/N): ";E$
3190  IF E$ < > "J" GOTO 240
3200  HOME
3210  GOTO 3060
3300  PRINT
3310 F1 = 1
3320  PRINT "ADRES MET SLEUTELWOORD:"
3330  PRINT "----------------------"
3340  PRINT "<";Z;">"
3350  PRINT N$(Z)
3360  PRINT A$(Z)
3370  PRINT P$(Z);"  ";W$(Z)
3380  PRINT "(";K$(Z);")"
3390  IF Z = N GOTO 240
3400  GOTO 3140
```

```
4000  HOME
4010  PRINT "**SORTEER**"
4020  PRINT "==========="
4030  PRINT
4040  PRINT "ER KAN GESORTEERD WORDEN OP ZOWEL        NAAM(=1), STRAAT(=2)
, POSTCODE(=3),       WOONPLAATS(=4) OF KENMERK(=5)"
4050  INPUT "UW KEUZE?(1,2,3,4 OF 5) : ";Q
4060  HOME : PRINT : PRINT : PRINT : PRINT : PRINT
4070  PRINT "SORTEREN KOST ZELFS EEN COMPUTER TIJD..."
4080  ON Q GOSUB 4100,4200,4300,4400,4500
4090  GOTO 4060
4100  FOR I = 0 TO N
4110  FOR J = 1 TO N
4120  IF N$(I) < N$(J) THEN  GOSUB 4600
4130  NEXT J
4140  NEXT I
4150  PRINT
4160  PRINT "BESTAND GESORTEERD OP NAAM"
4170  GOTO 240
4200  FOR I = 0 TO N
4210  FOR J = 1 TO N
4220  IF A$(I) < A$(J) THEN  GOSUB 4600
4230  NEXT J
4240  NEXT I
4250  PRINT
4260  PRINT "BESTAND GESORTEERD OP STRAAT"
4270  GOTO 240
```

```
4300  FOR I = 0 TO N
4310  FOR J = 1 TO N
4320  IF P$(I) < P$(J) THEN  GOSUB 4600
4330  NEXT J
4340  NEXT I
4350  PRINT
4360  PRINT "BESTAND GESORTEERD OP POSTCODE"
4370  GOTO 240
4400  FOR I = 0 TO N
4410  FOR J = 1 TO N
4420  IF W$(I) < W$(J) THEN  GOSUB 4600
4430  NEXT J
4440  NEXT I
4450  PRINT
4460  PRINT "BESTAND GESORTEERD OP WOONPLAATS"
4470  GOTO 240
4500  FOR I = 0 TO N
4510  FOR J = 1 TO N
4520  IF K$(I) < K$(J) THEN  GOSUB 4600
4530  NEXT J
4540  NEXT I
4550  PRINT
4560  PRINT "BESTAND GESORTEERD OP KENMERK"
4570  GOTO 240
```

```
4600 HN$ = N$(I)
4610 HA$ = A$(I)
4620 HP$ = P$(I)
4630 HW$ = W$(I)
4640 HK$ = K$(I)
4650 N$(I) = N$(J)
4660 A$(I) = A$(J)
4670 P$(I) = P$(J)
4680 W$(I) = W$(J)
4690 K$(I) = K$(J)
4700 N$(J) = HN$
4710 A$(J) = HA$
4720 P$(J) = HP$
4730 W$(J) = HW$
4740 K$(J) = HK$
4750  RETURN
```

```
5000  HOME
5010  PRINT "**VERANDER**"
5020  PRINT "=============="
5030  PRINT
5040  IF N > 0 GOTO 5070
5050  PRINT "ER IS GEEN BESTAND IN DE COMPUTER...."
5060  GOTO 240
5070  INPUT "WELK ADRESNR. WILT U VERANDEREN? : ";Z
5080  HOME
5090  PRINT "HET GEVRAAGDE ADRES WORDT NU REGEL"
5100  PRINT "NA REGEL GETOOND, ALS EEN REGEL OP HET"
5110  PRINT "SCHERM VERSCHIJNT KUNT U DE REGEL  "
5120  PRINT "CORRIGEREN DOOR HEM OPNIEUW IN TE VOEREN"
5130  PRINT "MOET DE REGEL BLIJVEN ZOALS GETOOND,"
5140  PRINT "TIK DAN TWEEMAAL HET $-TEKEN IN.($$)"
5150  PRINT
```

```
5160  PRINT N$(Z)
5170  INPUT "CORRECTIE? : ";V$
5180  IF  LEFT$ (V$,2) = "$$" GOTO 5200
5190 N$(Z) = V$
5200  PRINT A$(Z)
5210  INPUT "CORRECTIE? : ";V$
5220  IF  LEFT$ (V$,2) = "$$" GOTO 5240
5230 A$(Z) = V$
5240  PRINT P$(Z)
5250  INPUT "CORRECTIE? : ";V$
5260  IF  LEFT$ (V$,2) = "$$" GOTO 5280
5270 P$(Z) = V$
5280  PRINT W$(Z)
5290  INPUT "CORRECTIE? : ";V$
5300  IF  LEFT$ (V$,2) = "$$" GOTO 5320
5310 W$(Z) = V$
5320  PRINT K$(Z)
5330  INPUT "CORRECTIE? : ";V$
5340  IF  LEFT$ (V$,2) = "$$" GOTO 5360
5350 K$(Z) = V$
5360  PRINT
5370  INPUT "WILT U NOG EEN ADRES VERANDEREN?(J/N) : ";D$
5380  IF D$ = "N" GOTO 240
5390  HOME
5400  INPUT "WELK ADRESNR.? : ";Z
5410  GOTO 5150
```

```
6000  HOME
6010  PRINT "**LABELS**"
6020  PRINT "=========="
6030  PRINT
6040  INPUT "WILT U ETIKETTEN VAN ALLE ADRESSEN?(J/N) : ";G$
6050  IF G$ = "N" GOTO 6200
6060  GOSUB 6400
6070  PR# 2
6080  FOR E = 1 TO N
6090  PRINT : PRINT
6100  PRINT N$(E)
6110  PRINT A$(E)
6120  PRINT P$(E);"  ";W$(E)
6130  PRINT : PRINT : PRINT : PRINT
6140  NEXT E
6150  PR# 0
6160  GOTO 240
6200  HOME
6210  PRINT "U KUNT SELECTEREN OP ADRESKENMERK"
6220  INPUT "WAT IS KENMERK KEUZE? : ";H$
6230  GOSUB 6400
6240  PR# 2
6250  FOR K = 1 TO N
6260  IF K$(K) <  > H$ GOTO 6320
6270  PRINT : PRINT
6280  PRINT N$(K)
6290  PRINT A$(K)
6300  PRINT P$(K);"  ";W$(K)
6310  PRINT : PRINT : PRINT : PRINT
6320  NEXT K
6330  PR# 0
6340  GOTO 240
6400  HOME
```

```
6410  PRINT "ER WORDEN EERST TEST-ETIKETTEN AFGEDRUKTOM DE INSTELLING VAN
      DE PRINTER TE       CONTROLEREN"
6420  PR# 2
6430  PRINT : PRINT
6440  FOR D = 1 TO 3
6450  PRINT "XXXXXXXXXXXXXXXXXXXXXX"
6460  NEXT D
6470  PRINT : PRINT : PRINT : PRINT
6480  PR# 0
6490  INPUT "INSTELLING O.K.?(J/N) : ";I$
6500  IF I$ <  > "J" GOTO 6420
6510  RETURN
```

```
7000  HOME
7010  PRINT "**BEWAAR**"
7020  PRINT "=========="
7030  PRINT
7040  INPUT "GEEF NAAM WAARONDER BESTAND MOET WORDEN OPGEBORGEN : ";F$
7050 D$ =  CHR$ (4)
7060  PRINT D$;"OPEN";F$
7070  PRINT D$;"WRITE";F$
7080  PRINT N
7090  FOR L = 1 TO N
7100  PRINT N$(L)
7110  PRINT A$(L)
7120  PRINT P$(L)
7130  PRINT W$(L)
7140  PRINT K$(L)
7150  NEXT L
7160  PRINT "EOF"
7170  PRINT D$;"CLOSE";F$
7180  GOTO 240
8000  HOME
8010  PRINT "**EINDE**"
8020  PRINT "========="
8030  PRINT
8040  PRINT "HEEFT U HET ADRESBESTAND OPGEBORGEN??"
8050  INPUT "TIK J OF N IN : ";M$
8060  IF M$ <  > "J" THEN 240
8070  HOME
8080  VTAB (5)
8090  PRINT "DIT WAS DAN DUS........"
8100  PRINT
8110  PRINT "####################################"
8120  PRINT
8130  PRINT " ###  #     ###   # ####  #####"
8140  PRINT "#   # #     #   # # #   # #"
8150  PRINT "#     #     #   # # #   # #"
8160  PRINT "#     #     ##### # ####  ####"
8170  PRINT "#     #     #   # # # #   #"
8180  PRINT "#   # #     #   # # #  #  #"
8190  PRINT " ###  ##### #   # # #   # #####"
8200  PRINT
8210  PRINT "####################################"
8220  PRINT
8230  PRINT "EEN"
8240  PRINT "    ADRES"
8250  PRINT "            BESTAND"
8260  PRINT "                        PROGRAMMA"
8270  PRINT : PRINT
8280  PRINT "AUTEUR: C.W.DEN OTTER SR."
8290  PRINT "            ZES JANUARI 1980"
8300  END
```

```
9000  HOME
9010  PRINT "**HELP**"
9020  PRINT "============"
9030  PRINT
9040  GOTO 130
9050  END
```

Software routine voor de Exidy Sorcerer

Als besluit volgt als bijvoegsel een software routine geschreven voor de Exidy Sorcerer. Deze computer kan geen alfanumerieke arrays wegschrijven naar cassette.
Daarom moet een omzetting plaatsvinden van alfanumerieke naar numerieke arrays. Deze arrays kunnen wel naar cassette weggeschreven worden.
Deze informatie kan m.b.v. de routine van regel 200-370 weer worden ingelezen vanaf de cassette en worden terugvertaald naar alfanumerieke en dus (uit)leesbare arrays. Degene die dit supplement begrijpt zal in staat zijn de invoerroutine beginnende op regel 1000 en de bewaarroutine beginnende op regel 7000 dusdanig te modificeren dat dat het Claire programma voor de Exidy Sorcerer geschikt is.

```
1 REM SUPPLEMENT BEDOELD VOOR EXIDY GEBRUIKERS
2 REM DIT PROGRAMMA ZET ALFANUMERIEKE STRINGS OM
3 REM NAAR EEN NUMERIEK ARRAY WAT DAARNA NAAR
4 REM CASSETTE WEGGESCHREVEN WORDT.
5 REM DIT PRINCIPE KAN WORDEN GEBRUIKT VOOR HET
6 REM VERANDEREN VAN DE VOOR EEN ANDERE COMPUTER
7 REM GESCHREVEN INLEES- EN BEWAARROUTINE VAN HET
8 REM CLAIRE PROGRAMMA.
10 CLEAR 30000
20 INPUT"NAAM: ";N$
30 INPUT"STRAAT: ";A$
40 INPUT"WOONPLAATS: ";W$
50 DIMB(30,3)
60 T$=N$
70 W=1
80 GOSUB1000
90 T$=A$
100 W=2
110 GOSUB1000
120 T$=W$
130 W=3
140 GOSUB1000
150 CSAVE* 1 B
200 INPUT"TERUGGESPOELD ";Q$
210 IFQ$<>"J"GOTO200
220 DIMD(30,3)
230 CLOAD* 1 D
240 W=1
250 GOSUB2000
260 N1$=D$
270 W=2
280 GOSUB2000
290 A1$=D$
300 W=3
310 GOSUB2000
320 W1$=D$
330 PRINTCHR$(12)
340 PRINTN1$
350 PRINTA1$
360 PRINTW1$
370 END
1000 L=LEN(T$)
1010 B(0,W)=L
1020 FORR=1TOL
1030 B(R,W)=ASC(MID$(T$,R,1))
1040 NEXTR
1050 RETURN
2000 K=D(0,W)
2005 D$=""
2010 FORJ=1TOK
2020 D$=D$+CHR$(D(J,W))
2030 NEXTJ
2040 RETURN
READY
```